修 士 論 文

マルチコア上での
ソフトウェアフォルトトレランス技術に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

川口 貴司

2010 年 3 月

修 士 論 文

マルチコア上での
ソフトウェアフォルトトレランス技術に関する研究

指導教員　Defago Xavier 准教授

審査委員主査　Defago Xavier 准教授
審査委員　青木利晃 准教授
審査委員　岸知二 客員教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

0810015 川口 貴司

提出年月: 2010 年 2 月

概 要

組み込みソフトウェアが，あらゆるものに組み込まれ，その信頼性は一層重要になってきている．機器の高信頼化の施策は一般的に高コストであり，開発コストとの兼ね合いから，民生品では施策しづらかった．一方で，高性能化や高機能化要求に対する技術として，組み込み向けマルチコアチップが実用化され，今後マルチコアを活用した組み込みソフトウェアの高性能化，高機能化が期待されている．

本研究はマルチコアを利用して，民生品レベルの組み込み機器を，安価かつ手軽に高信頼化することを目的とする．具体的なアプローチとして，マルチコアを利用したプロセスペアの方式を提案するとともに，提案方式に沿ったアプリケーション開発を支援するためのフレームワークを実現した．

目次

第1章　はじめに

1.1　背景

組み込みソフトウェアが組み込まれた機器領域の分類を図 1.1 に示す (特許庁『技術分野特許マップ』[1] より引用)．社会インフラや産業を支える機器，生活家電に至るまで，非常に多岐にわたる．組み込みソフトウェアは，あらゆるものに組み込まれ，その信頼性は一層重要になってきている．また近年，こうした信頼性に対する消費者の関心は，図中の枠内のような身近な民生品に対しても高まり，開発者の意識も高まっている．しかしながら，信頼性を向上させるための施策は一般的に高コストであり，コスト制約の厳しい民生品にそのまま適用することは困難である．したがって，民生品に見合ったコストで実現可能な高信頼化の施策が必要とされている．

高信頼化に有効な施策の一つとして，従来からフォルトトレランス技術がある．人命にかかわるシステムや社会インフラのような大規模なシステムでは，高信頼システムであることが必須であり，多大なコストをかけた高信頼化の施策がなされてきた．こうした施策には，ハードウェアの冗長構成や，ソフトウェアフォルトトレランス機能の実装が必要であり，開発コストとの兼ね合いから民生品では施策しづらい．一方で，高性能化や高機能化要求に対する技術として，現在注目されているのが一つのチップにＣＰＵコアを複数搭載したマルチコアチップである．各ＣＰＵコアへ機能分散することで，システムの高性能化と省電力化を同時に実現可能であり，すでにパソコンでは一般的に利用されている．近年，組み込み機器向けのマルチコアチップが実用化された．マルチコアを応用した高性能化に関する研究は盛んに行われており，マルチコア用に組み込みソフトウェアを最適化するツールやコンパイラも登場予定である．今後，マルチコアを活用した組み込みソフトウェアの高性能化が期待されている．

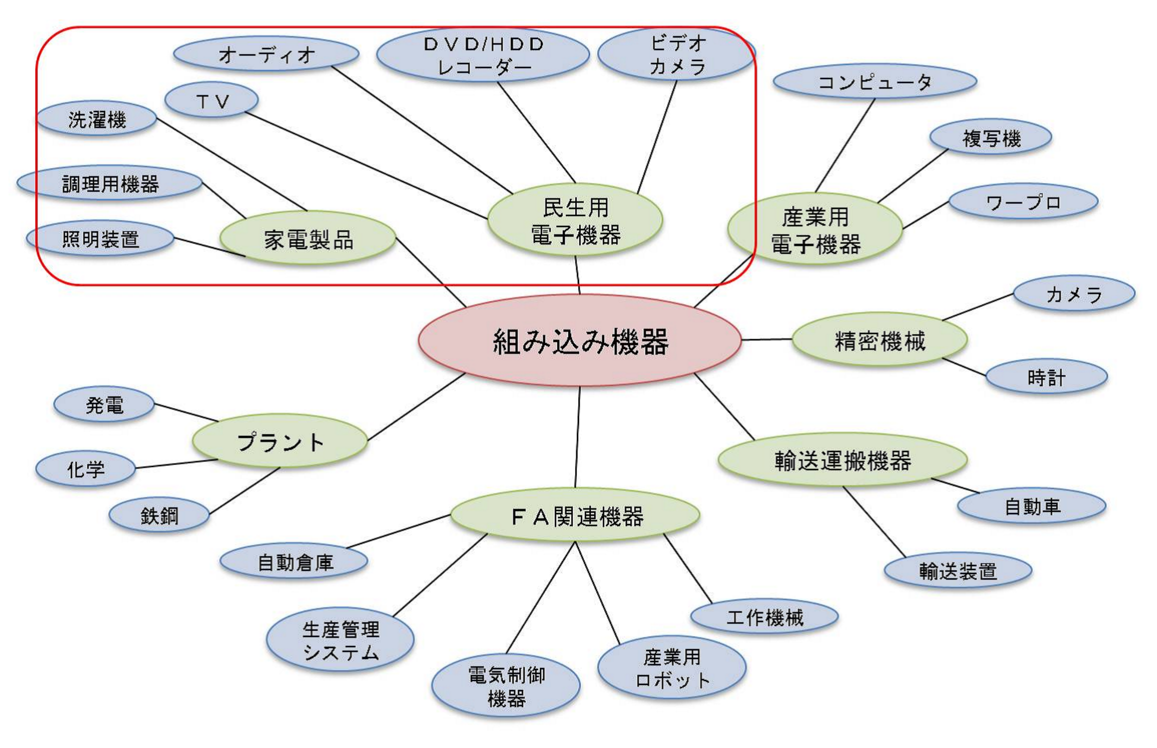


図 1.1：組み込み機器領域の分類

1.2　目的

本研究では，従来主として高性能化目的に用いられてきたマルチコアを，高信頼化目的に利用する．本研究の目的は，マルチコアを用いて，民生品レベルの組み込み機器を安価かつ手軽に高信頼化することである．具体的なアプローチとして，マルチコアを利用したプロセスペアの方式を提案するとともに，提案方式に沿ったアプリケーション開発を支援するためのフレームワークを実現する．

1.3　本論文の構成

本論文の構成は以下のとおりである．

2 章では本研究で取り扱った既存技術の概要について述べ，3 章で本研究が解決を目指す課題について述べる．4 章では，本アプローチの概要について述べる．本アプローチについて提案方式とフレームワークに分けて，それぞれ 5 章と 6 章にて詳しく述べる．7 章ではそれらの評価について述べる．8 章では研究のまとめと今後の課題について述べる．

第2章　既存技術

本章では，本研究において重要な要素である各既存技術について説明する．

2.1　フォールトトレランス技術について

2.1.1　フォールトトレランス技術

正常時のシステムがユーザに与える出力をサービスと呼ぶ．信頼性とはシステムがサービスをどれだけの期間，提供し続けるかを示す品質特性であり，一般的には，サービスを提供し続けた時間の平均をその尺度とする [2]．フォールトトレランス技術とは，稼働中のシステムでフォールトが発生しても，サービスを継続するための技術の総称である．フォールトとはシステム構成要素内で発生する故障である．フォールトの発生によって引き起こされるシステム構成要素内の状態の異常をエラーと呼び，エラーによって最終的に引き起こされるサービスの異常をシステム障害と呼ぶ．通常，フォールトの存在と発生は区別される．ソフトウェアの場合の一例をあげると，ソフトウェアの典型的なフォールトとしてはバグがある．フォールトの存在とは，プログラムにバグが含まれていることで，バグを含むプログラムコードが実行されることがフォールトの発生といえる．そして実行されたことによって，プログラムの関数が無限ループ状態に陥ることなどがエラーといえる．

Laprie らは種々の研究をまとめ，フォールトの諸特性に基づき，その分類を図 2.1 のように定義している [3]．フォールトトレランス技術は，フォールトの存在は避けられないものと考え，こうしたフォールトの分類から，システムに存在すると考えられるフォールトがもつ特性を考慮し，フォールトが発生した場合でも，システム障害の発生を防ぐ，もしくは軽微に抑えることが目的であり，フォールトに耐える性能（耐障害性）を実現するための技術である．

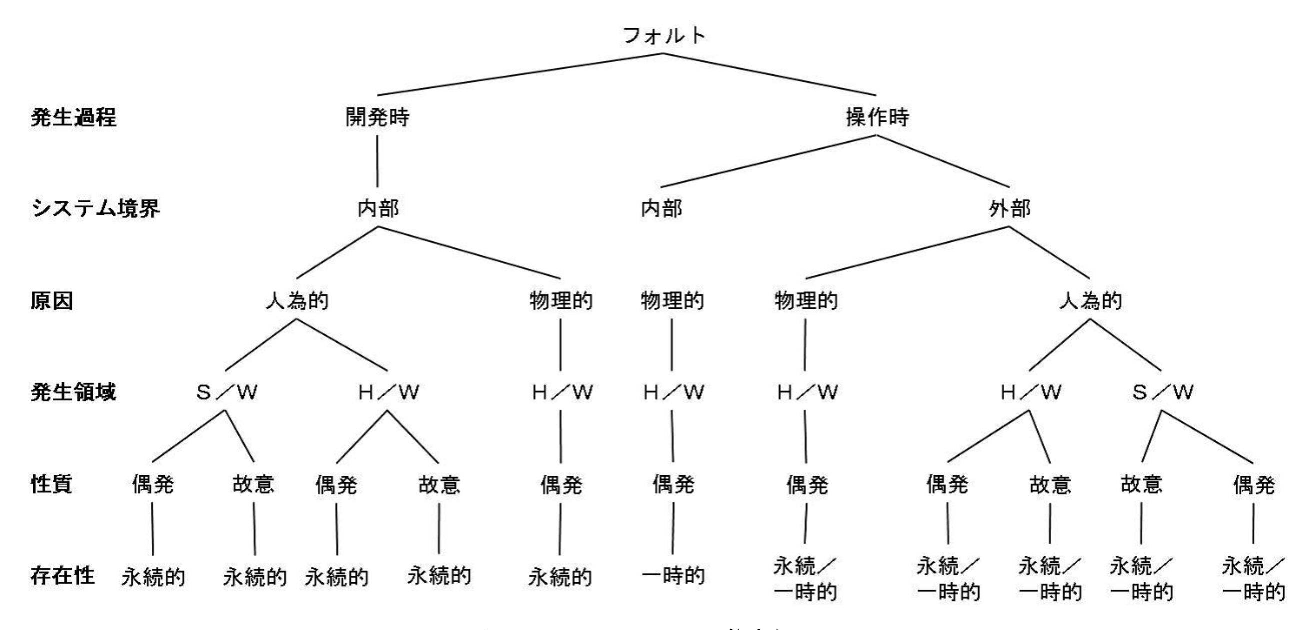


図 2.1：フォルトの分類

2.1.2　耐障害性レベル

Laprie らは，システム障害は３種類の観点によって性格づけられるとしている．Laprie らが，障害発生領域の観点から分類し，定義しているシステム障害の代表例を表 2.1 に示す [4]．いずれの障害も一時的に発生する場合もあれば，永続的に発生する場合もある．システムに求められる信頼性，および，信頼性の副特性である障害許容性や回復性などの品質（ISO/IEC9126[5] による定義) に基づき，必要とされるフォルトトレランス技術のレベルは様々である．藤原ら［6］はユーザに対する障害の見え方によって，フォルトトレランス技術のレベルを表 2.2 に示す３つのレベルに分類している．本論文でもこの分類を参考にしている．

表 2.1：システム障害の代表例

障害の種類	概要
値の障害（value failures）	サービスの内容が仕様と一致しない
タイミング障害（timing failures）	サービスのタイミングが仕様と一致しない
停止障害（stopping failures）	上記二つの障害が同時に起こる
非稼働障害（omission failure）	サービスがまったく提供されない （停止障害の特殊系と位置付けられている）

表 2.2：フォルトトレランス技術のレベル

レベル	概要
動的マスクレベル	障害を一切生じさせないレベル． ユーザはフォルトやエラーの発生に気づくことはない
静的マスクレベル	一時的に障害を引き起こすが，自動的に回復する． ユーザは障害に気づくときもあれば気づかない時もある
フェイルセイフ	障害は発生するが，その影響を設計者の意図した範囲に収める（常に安全側の影響しか生じない）

2.1.3 ソフトウェアフォルトトレランス技術

フォルトトレランス技術のうち，ソフトウェアのフォルトの発生に備えるものを特にソフトウェアフォルトトレランス技術と呼ぶ．ソフトウェアフォルトの多くは，プログラムのバグである．バグは，それ自体が障害を引き起こすものと，外部から別のフォルトが混入される原因（例えばセキュリティホール）になるもの等様々である．ソフトウェアフォルトトレランス技術は基本的に，こうしたフォルトの発生（エラー）を検知するための機能とシステムを回復させるための機能の組み合わせで考えることができる．代表的なエラー検知機能を表 2.3，システム回復機能を表 2.4 に示す．

表 2.3：代表的なエラー検知機能

エラー検知機能	概要
ウォッチドッグタイマ（ＷＤＴ）	ウォッチドッグタイマのカウント値が上限値を超えたら，エラーと判定する
受入れテスト	処理結果を，エラー判定条件と比較する
出力比較	複数のソフトウェアの処理結果を比較する

表 2.4：代表的なシステム回復機能

回復機能	概要
リセット	ソフトウェアまたはハードウェアリセットを行う
代替処理	予め決めておいた別処理を行う
切替処理	別のＣＰＵや，ソフトウェアに切り替える
チェックポイント＆ロールバック回復	チェックポイント箇所から再開する

これらを組み合わせた代表的な技法として，同じソフトウェアを異なるＣＰＵで稼働させ，エラー検知時に処理ＣＰＵを切り替えることでサービスを継続するプロセスペア (2.2 に詳細記述) や，異なる設計のソフトウェアに処理を切り替えるリカバリブロック等がある．

フォルトトレランス技術の導入例を表2.5に示す[7]．ここで取り上げた例からわかるように，ハードウェアの冗長構成と，ソフトウェアフォルトトレランス機能を組み合わせることで，より高度なレベルの耐障害性を実現可能である．

表2.5：フォルトトレランス技術の導入例

導入先システム	概要
スペースシャトルのメインコンピュータ	**5重系**：ハードウェアとソフトウェアを４重化し多数決＋１つのバックアップ（最悪でも４重系のうち，２つの故障まで耐えられる）
飛行機の制御システム	**3×3重系**：異なるバージョン（ＯＳ・設計）からなる，３重系を３つのネットワークで接続
銀行のトランザクションシステム	**2重系（プロセスペア）**：ハードウェアとソフトウェアをそれぞれ２重化し，稼働中のシステムがダウンすると予備系統が処理を引き継ぐ
車のエンジン制御システム	**フェイルセイフ**：エンジン点火中にシステムが停止すると予備システムがエンジンを安全に停止，もしくは低出力で制御する

2.1.4　フォルトトレランス技術の問題点

2.1.3で紹介した導入例は，いずれもハードウェアとソフトウェア両方の導入コストが高い．特にスペースシャトルと飛行機の例は典型的な静的レベルのフォルトトレランス技術であり，開発費用，開発期間ともに非常に大きい．ソフトウェアフォルトだけを対象とするのであれば，ハードウェアの冗長化は行わず，ソフトウェアフォルトトレランス機能を作り込むことで，ハードウェアコストを抑え，高信頼化できる可能性はある．しかしながら，ソフトウェアコストの増加と，ソフトウェアの複雑化は避けられない．ソフトウェアの複雑化によって，新たなソフトウェアフォルトを生じさせる可能性もある．当然ながら，ソフトウェア開発者の負担も増加する．フォルトトレランス技術の導入を検討する際は，基本的にシステムの高信頼化の開発費用や開発期間と，高信頼性の必要度合いとのトレードオフである．また組み込み機器の場合は物理的な大きさが問題となることもある．

2.2　プロセスペアについて

1.2で述べたように，本研究では，マルチコアを使用したプロセスペアの方式を提案する．この節では，一般的なプロセスペアの概要を説明する．

プロセスペア（待機冗長システムやデュアルシステムとも呼ばれる）は，典型的な動的マスクレベルのフォルトトレランス技術である．その基本的な動作構造を図2.2に示す．同じソフトウェアをメインＣＰＵとサブＣＰＵの二つの異なるＣＰＵで稼働させる．正常

時はメインＣＰＵのみで処理を行う．メインＣＰＵに何らかの障害が発生すると，サブＣＰＵに切り替わり処理を継続する．メインＣＰＵが正常に稼働している時のサブＣＰＵの状態は表 2.6 に示す３つに分類される [8]．またエラー検知をメインＣＰＵ自身が行う，もしくはサブＣＰＵや他のシステムに行わせるなど，プロセスペアに必要な機能の実装方式によってソフトウェア構造は異なる．プロセスペアの導入にあたり，一般的に必要な検討事項と典型的な実装方式例を図 2.3 に示す．図中 2 のエラー検知に関しては，メインＣＰＵ自身が自らのエラーチェックを行う自己診断方式，サブＣＰＵがメインＣＰＵの状態を監視する方式，外部のシステムに監視させる方式が考えられる．図中 3 のシステム回復に関しては，メインＣＰＵとサブＣＰＵの状態を常に同期させておき瞬時に切り替える方式，エラーが発生する前のシステム状態まで戻る方式，初期状態から再開する方式などがある．図中 4 のＣＰＵ切替方式は，メインＣＰＵが，自身に障害が発生したことをサブＣＰＵに通知し，自身を停止させる方式，サブＣＰＵがメインＣＰＵを停止させる方式，ＣＰＵ外部のシステムが行う方式が考えられる. 図中の方式を組み合わせ，その方式を実現可能なハードウェア環境とそれに合わせたソフトウェアの実装が必要となる．

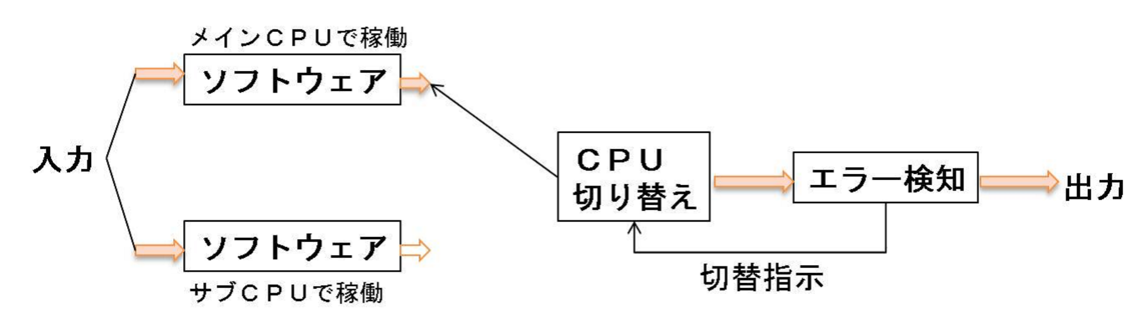


図 2.2：プロセスペアの基本動作構造

表 2.6：サブＣＰＵの状態

名称	概要
コールドスタンバイ	電源が切れている状態
ウォームスタンバイ	コールドスタンバイとホットスタンバイの中間
ホットスタンバイ	電源が入り，メインＣＰＵと同期

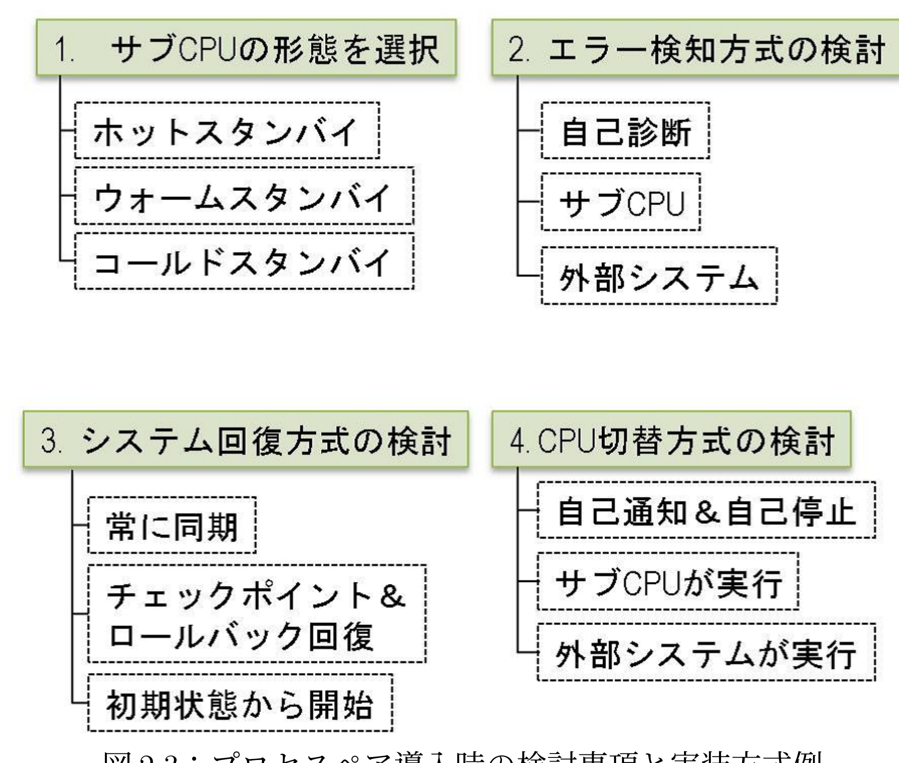


図 2.3：プロセスペア導入時の検討事項と実装方式例

2.3　マルチコアについて

2.3.1　マルチコア

マルチコアとは，一つのチップに複数のＣＰＵコアが搭載されたものである．一般的な活用方法は，複数のＣＰＵコアにシステムの機能を分散し並列処理させる．こうすることで，システムの高性能化と消費電力の抑制が同時に可能である．汎用コンピュータの世界では，ＣＰＵにマルチコアチップを採用し，高性能化する事が当たり前となっている．近年の組み込み業界でも，高まり続ける高性能化のニーズに対する有効な技術として開発が進められ，組み込み機器向けのマルチコアチップが台頭してきた．

2.3.2　マルチコアの特徴と分類

組み込み向けマルチコアチップには，コア間のバスや，共有メモリも予め搭載されており，従来複数のＣＰＵチップとハードウェア部品が必要であったマルチプロセッサ環境が一つのチップで構築されている．

一般的にマルチコアチップは，搭載するＣＰＵコアの種類とＯＳの使い方から，ＳＭＰ（Symmetric Multiprocessing）とＡＭＰ（Asymmetric Multiprocessing）の２種類に大別される (図 2.4)[9]．本研究で扱うチップは，一つのチップにＣＰＵ＃０とＣＰＵ＃１と

いう二つのＣＰＵコアが搭載され，ＡＭＰ構成が採用されたデュアルコアチップである．図 2.5 にイメージ図を示す．共有メモリに各ＣＰＵコアの専用領域と，共有領域を持ち，メモリ間の排他制御を行うためのセマフォ・レジスタや，ＣＰＵ間割り込みコントローラ（どちらもハードウェアレベルでサポート）が搭載されている．こうしたマルチプロセッサ環境を支援する専用ハードウェアが予め搭載されている場合が多いのもマルチコアチップの特徴といえる．

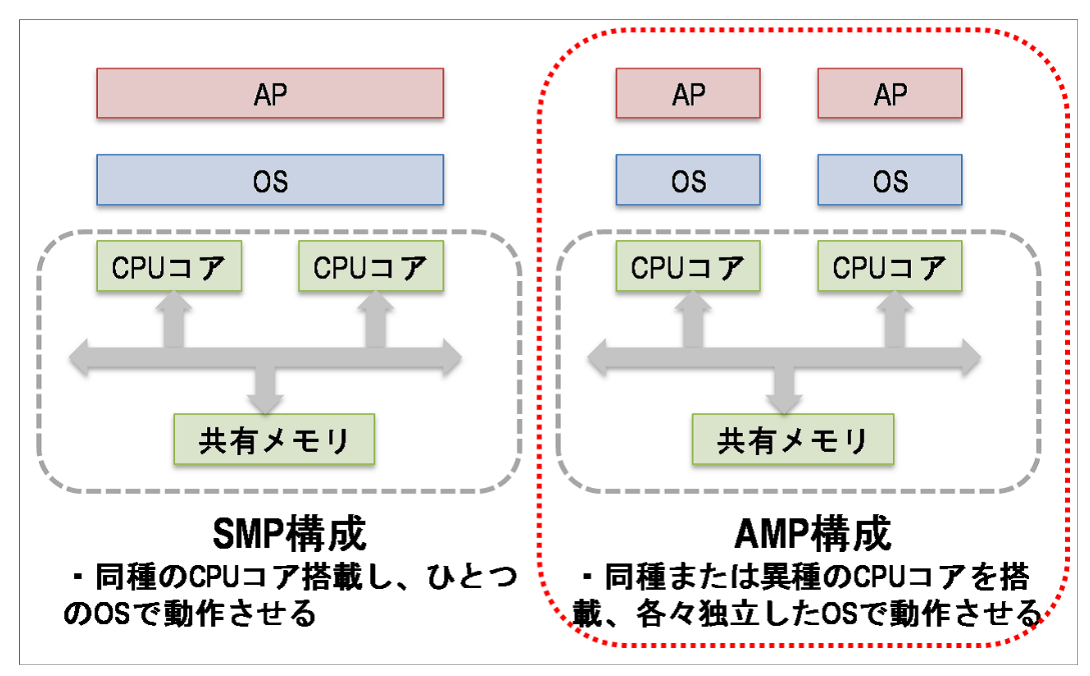


図 2.4：マルチコアの分類

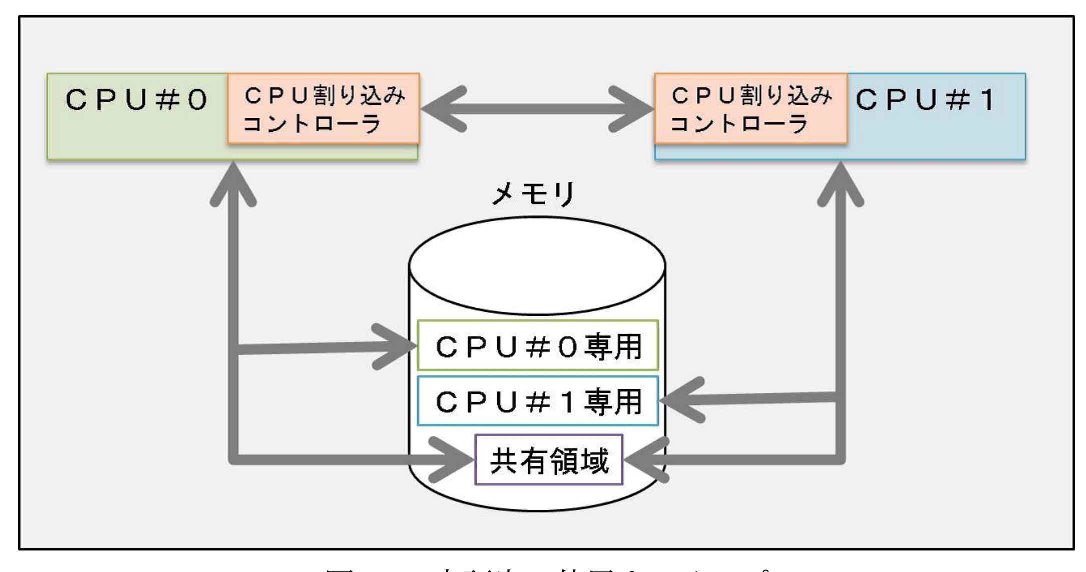


図 2.5：本研究で使用するチップ

第3章　課題

本章では，本研究で対象とする民生品レベルの機器に求められる信頼性と，解決しようとする課題について述べる

3.1　求められる信頼性

従来，高信頼化の施策がなされてきたシステムと，民生品レベルの機器との特性の違いから，求められる信頼性のレベルについて述べる．従来，高信頼化の施策がなされてきたシステム（2.1.3 で示した導入例など）の場合，システムにひとたび障害が発生すれば被害は甚大であり人命危機や社会的混乱に陥る可能性が非常に高い．ゆえに高信頼システムであることが必須であり，その特性上，多くの場合は静的マスクレベルのフォルトトレランス技術が必要とされる．一方，民生品レベルの機器の場合は，仮に障害が発生しても，個々の被害の重大度は小さく人命に関わる可能性は低い．そのため，高信頼化の必要性がありつつも，開発費と開発期間の制約上，高信頼化の施策は見送られることが多かった．しかしながら近年，前述したように，組み込みソフトウェア領域の拡大と，こうした民生品レベルの機器の信頼性に対して，消費者の関心が高まったことで，高信頼化の施策の必要性は一層高まった．機器の特性上，前者のシステムのように，静的マスクレベルの信頼性が必須とはいえないまでも，民生品レベルの機器に見合ったレベルの信頼性（動的マスクレベル）の施策が必要とされている．

3.2　課題

高信頼化の必要性が高まる一方で，従来から組み込み機器の開発制約は，厳しいままである．特に民生品レベルの機器では，製品の多様化，機能の複合化，新製品登場期間の短期化により，機器の開発費・開発期間の制約は従来よりも増加している．

発生するソフトウェアフォルトの形態も変化してきている．従来の民生品には，シンプルな機能とユーザインターフェースしかなく，動作も単純なものであった．しかしながら，近年の種々の技術発展と市場からの要求により，多機能化し，入力スイッチや表示ディスプレイ等のユーザインターフェースも充実し，ソフトウェアシステムが複雑化した．その結果，システムの内部状態とシステム外部イベントとの組み合わせにより不具合が発生した事例が増加傾向にある．下記にメーカーが発表した不具合の実例を紹介する．

実例１：携帯電話

メール作成中（キー操作中）に，突然画面がフリーズし，キー操作を受けつけなくなる

実例２：ＤＶＤプレーヤー

特定地域でワンセグ放送受信中に，メニュー項目を選択すると，動作が止まる場合がある

実例３：オーディオプレーヤー

音楽ファイルを再生しようとした時に，稀に機器が操作できなくなり，その後自動的に電源が切れて再度入る

こうした不具合に対して，短期間に全ての使用状況を予測し，備えることは困難である．ソフトウェアシステムの複雑化は，それ自体の開発だけでもソフトウェア開発者の負担を増加させている．そのため，ソフトウェアフォルトトレランス機能の開発に注力することは難しい．

本研究では，下記に示した点を，民生品レベルの機器に対して，高信頼化の施策を行う上での解決すべき主課題として捉えた．

- より低コストで実現可能な高信頼化の施策が望まれること
- 施策のための開発者の負担を減らすことが望まれること

第4章　アプローチ

1.1 で述べたとおり近年，組み込み機器向けのマルチコアチップが実用化され，それを利用した機器の高性能化に関する研究が盛んである．本研究では，マルチコアを組み込み機器の高信頼化を目的に利用する．本章では，本研究のアプローチの概要を次の２点に分けて説明する．

- マルチコアを使用したプロセスペアの方式の提案について
- 提案方式の導入支援フレームワークの実現について

4.1　マルチコアを使用したプロセスペアの方式の提案

マルチコアの特徴を活かすソフトウェアフォルトトレランス技術という観点から，プロセスペアを採用した．

マルチコアの最大の特徴は，一つのチップでマルチプロセッサ環境が構築されていることである．この特徴を活かすソフトウェアフォルトトレランス技術として，プロセスペアを取り上げる．2.2 で述べたとおり，プロセスペアの実装にはマルチプロセッサ環境が必要であり，シングルコアチップ単体では実現不可能である．従来，マルチプロセッサ環境を構築するには，ＣＰＵチップおよび，その他のハードウェア部品（バスや共有メモリ等）の増加を伴い，当然ながら開発コストの増加と構築環境の大型化につながる．そのため，生活家電やテレビ・オーディオなど身近な組み込み機器のように，特に小型化・低価格が望まれる製品に実装されることはなかった．しかし，マルチコアチップを用いることで，プロセスペアの各機能を一つのチップに実装し，ハードウェアコストを抑え，従来よりも安価かつコンパクトに実現することが可能である．なおかつソフトウェアから見た特性は，従来のマルチプロセッサ環境と大差のないプロセスペアの実現が可能である．後述の章（５章）にて，具体的な方式を検討し，検討に基づき，マルチコアを使用したプロセスペアの方式（以降，マルチコア用プロセスペア方式と呼ぶ）を提案する．また，マルチコア用プロセスペアという耐障害性モデルはまだ存在していない．本論文に後述する評価（７章）において，マルチコア用プロセスペアの有用性についても述べる．

4.2　フレームワークの実現

マルチコアに適した開発支援の手法という観点から，フレームワークを採用した．

開発期間や開発工員の制約が厳しい民生品の開発においては，本方式を利用するための開発者の負担を減らすことが望まれる．そこで提案方式の導入を支援するフレームワークを実現する．組み込みソフトウェアの構造は基本的に組み込む対象のハードウェアに依存する．マルチプロセッサ環境の場合，ソフトウェア構造は，構成要素であるＣＰＵチップやチップ間のバス，共有メモリ等の種類ごとに依存関係があり，各ハードウェアの接続の仕方にも依存する．当然ながら，構成ハードウェアの点数に比例し，ソフトウェアコストも増加する．しかしながら，一つのチップでマルチプロセッサ環境が構築されているマルチコアチップであれば，マルチプロセッサ環境用のソフトウェア構造でも（すなわちプロセスペアのソフトウェア構造でも），依存関係を一つのチップ内で収めることが可能である．そのため，ソフトウェアフレームワークとの相性が良く，適した支援手法であると考察した．ＣＰＵ間割り込みコントローラやセマフォ・レジスタなどのマルチプロセッサ環境をサポートする専用ハードウェアが搭載されている点も，プロセスペアの機能から見て相性が良い．プロセスペアの基本機能を，組み込み向けマルチコアチップに一般的に搭載されている専用ハードウェアを利用して実現し，そのソフトウェアフレームワークを実現することで，他社のマルチコアチップに変更しても再利用性の高いフレームワークが望める．フレームワークの構造の詳細については，６章にて説明する．

4.3　アプローチの前提条件

耐障害性を考える上で，フォルトモデル（フォルトの存在位置や発生条件などの仮定）を明確にすることが重要である．フォルトの存在位置（例えば，メイン関数や他の関数，割り込みハンドラに存在するなどの仮定）は，ソフトウェア構造に依存する．代表的な組み込みソフトウェア構造を表 4.1 に示す [10]．民生品レベルの機器にはイベント駆動型アーキテクチャが採用されていることが多い．本論文では，時間駆動型も含め，ＲＴＯＳなしアーキテクチャを主対象とする．また，実現するフォルトトレランス技術のレベルは動的レベルとし，ハードウェアのフォルトは対象としていない．本論文では，この前提に基づき，方式の検討を行った．なお，最終的に対象となるフォルトモデルはフレームワークを適用するアプリケーションに依るため本論文では議論しない．

表 4.1：代表的な組み込み向けソフトウェア構造

<table>
<tr><th colspan="2">ソフトウェア構造</th><th>概要</th></tr>
<tr><td colspan="2">ＲＴＯＳベース</td><td>ＲＴＯＳを中心とした構造</td></tr>
<tr><td rowspan="2">ＲＴＯＳなし</td><td>イベント駆動型</td><td>・main 関数と，主に割り込みハンドラから構成される
・民生品レベルの機器でよく採用される</td></tr>
<tr><td>時間駆動型</td><td>・main 関数とタイマ割り込みハンドラのみで構成される
・ハードリアルタイムシステムに向いている</td></tr>
</table>

4.4　関連研究

マルチコアを用いたシステムの高信頼化という観点で，関連した研究を紹介する．

- 組み込みＯＳ間の資源の共有やメモリ保護に関した研究（筑波大学）[11]

  ：主にコア間のメモリ保護，排他制御の効率化を支援する組み込みＯＳの研究

- 通信端末向け組み込みソフトウェアを対象とした，ネットワーク上からのフォルトの混入や被害を抑える研究・製品（ＮＥＣ）[12]

  ：通信先の信用度に応じて，入出力処理を行うコアを切り替える研究

マルチコアを利用した組み込み機器の高信頼化に関する研究自体少なく，本アプローチのようにマルチコアとプロセスペアに観点をおいた研究は未開領域である．

第5章　マルチコア用プロセスペアの提案

従来のプロセスペアの実装方式との比較から本方式の要点について説明した後，マルチコア用プロセスペアの方式と実現構造を提案する．

5.1　本方式の要点

従来の一般的なプロセスペアの実現方式と本方式について，アーキテクチャごとの比較を表5.1に示す．表からも分かるとおり，本方式の最大の要点は，従来ハードウェア単位で行っていたＣＰＵの切り替え動作を，同じチップ内のＣＰＵコアを切り替えることで処理を切り替える点である．以降本論文でのＣＰＵ切替とはコアの切替を指す．また，この切替機能を始め，従来複数のハードウェアが必要であったプロセスペアを本方式では，プロセスペアに必要なソフトウェア構造をすべて一つのチップに実装した．プロセスペアをマルチコアチップ一つで実現している点も要点である．

表5.1：従来方式と本方式の比較

	従来方式	**本方式**
ハードウェア	ＣＰＵ，入出力装置，バス等 ほぼ全てのハードウェアが 完全に２重化されている	**・ＣＰＵのコアが２重化されている** ・入出力装置は一つ ・バスは専用と共通のものがある
ソフトウェア	・一つのソフトウェアをＯＳが管理し別々のＣＰＵで稼働，または，ソフトウェアも２重化し両方のＣＰＵで稼働させる ・ソフトウェアは複数のハードウェアに分散する	・ソフトウェアを２重化して， **異なるコアに実装する** ・一つのチップにまとまる
耐障害性構造 （ＣＰＵの切替構造）	ハードウェアレベルで同期し 障害時は，片系を切り離して サービスを継続する	**コアを切り替えて**，障害発生前の 状態から，サービスを再開する

5.2　プロセスペアの方式の提案

提案方式を 2.2 で示したプロセスペアの導入時の検討事項に沿って説明する．なお方式検討の際は，下記の点を考慮した．

- 一つのマルチコアチップで実現可能であること
- エラーの種類やアプリケーションに依らず，ある程度汎用的に用いることが可能であること
- アプリケーションプログラムにプロセスペアのための仕組みを作り込む必要が少ないこと

1. サブＣＰＵの形態

ウォームスタンバイ方式を提案する．

マルチコアチップの特性上，コールドスタンバイは実現不可能である（両方のＣＰＵが同時に起動するため）．メインＣＰＵの異常発生に備えるサブＣＰＵの形態としてウォームスタンバイとホットスタンバイを比較し，ウォームスタンバイを選択した．選択理由を下記に示す．なお，本研究ではウォームスタンバイを“ ハードウェアは起動しているが，ソフトウェアは停止している状態 ”と定義する．

- ホットスタンバイの場合は，アプリケーションプログラムに同期をとる仕組みを作り込む必要があり，開発者の負担が増える. ウォームスタンバイであれば，より手軽に利用可能
- 動的マスクレベルの耐障害性であれば，ウォームスタンバイで実現可能
- ホットスタンバイより省電力

2. エラー検知方式

自己診断方式を提案する．

前節で，ウォームスタンバイ（ソフトウェアは停止している）を採用したため，エラー検知をサブＣＰＵに行わせることは不可能である．プロセスペアをマルチコアチップ一つで実現するために，エラー検知をチップ外部に実装する方式は避け，メインＣＰＵ自身が，自らのエラーチェックを行う自己診断方式を採用した.

3. システム回復方式

チェックポイント＆ロールバック回復方式を提案する．

チェックポイント＆ロールバック回復は，システム回復方式として一般的に用いられている方式である．この方式の基本的な動作を図 5.1 に示す．メインＣＰＵは一定間隔で，システム状態を記憶装置に保存する（この動作をチェックポインティング，保存されるシステム状態はチェックポイントと呼ばれる）．メインＣＰＵに障害が発生すると，最新のチェックポイントを利用してシステム状態を回復し，サブＣＰＵが処理を再開する．チェックポイントは両方のＣＰＵからアクセス可能なチップ外部の共有メモリに保存される場合が多い．本論文では，チップ内に搭載されている両ＣＰＵコアから共通アクセス可能なメモリにチェックポイントを保存する方式を提案する．システム外部に共有メモリを用意する必要が無く，一つのチップで実現可能である．

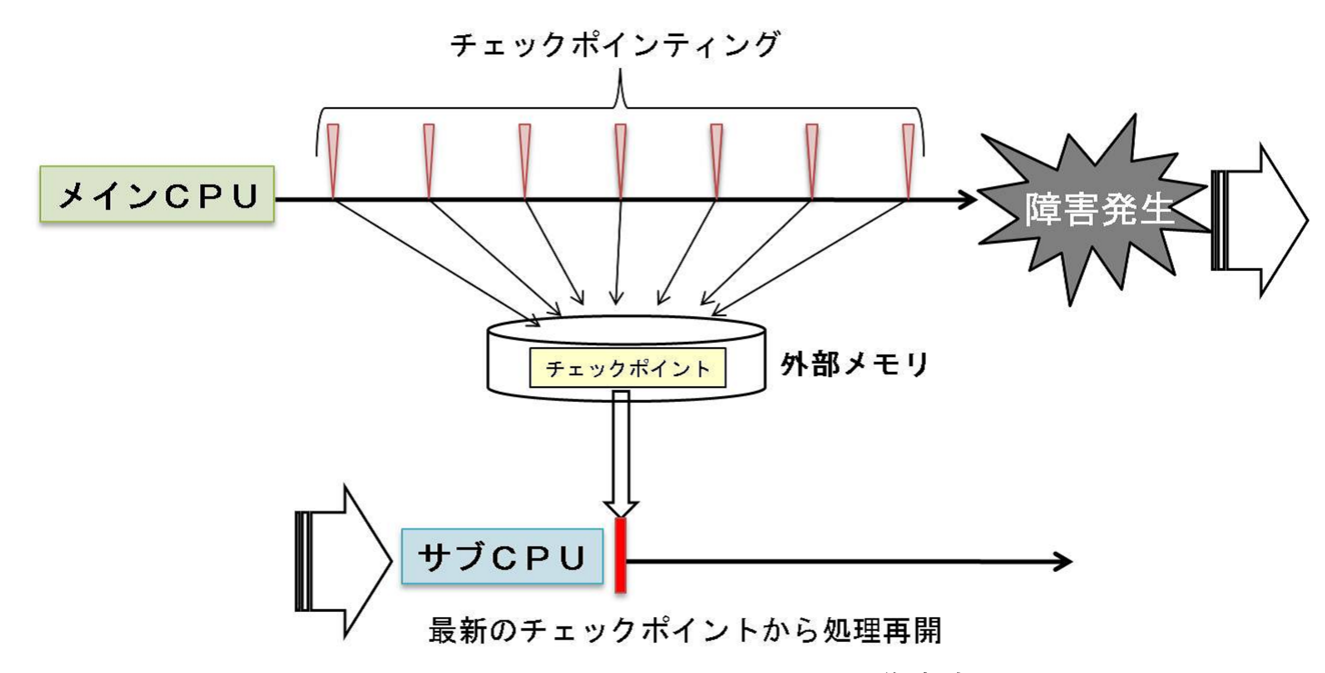


図 5.1：チェックポイント＆ロールバック回復方式

4. ＣＰＵ（コア）切替方式

自己通知＆自己停止方式を提案する．

ＣＰＵ（コア）切替方式に関しても，プロセスペアをマルチコアチップ一つで実現するために，自己通知＆自己停止方式を採用した. すなわち，エラーが検知された場合，メインＣＰＵ自らが，自身に異常が起きたことをサブＣＰＵに通知し，自身を停止させる．サブＣＰＵはメインＣＰＵからの通知を受け，ウォームスタンバイを解除し，処理を再開することでＣＰＵが切り替わる方式を採る．

5.3　ソフトウェア構造の提案

5.2 で提案したプロセスペアの方式を実現するソフトウェア構造を提案する．提案方式に基づくプロセスペアの動作フローについて説明した後，実現ソフトウェア構造について説明する．

提案方式に基づくプロセスペアの動作

提案したプロセスペアの方式に基づくメインＣＰＵとサブＣＰＵの各動作フローの一般形を図 5.2 に示す．図中，赤色の動作がプロセスペアに関する動作である．本研究で用いるチップにはＣＰＵ＃０と＃１という二つのＣＰＵコアが搭載されている．両コアに同一のアプリケーションプログラムを実装する．ハードウェア起動後，ＣＰＵ＃０がメインＣＰＵとなりアプリケーションプログラムを実行する．ＣＰＵ＃１がサブＣＰＵを務め，ウォームスタンバイ状態となり，メインＣＰＵの障害発生に備える．エラー発生時の誤ったシステム状態がチェックポイントとならないように，エラーの自己診断を先に行い，エラーが無いと判断された場合にのみチェックポインティングを行う．なおこの動作フローは一般形であって，アプリケーションプログラムの構造や，エラー検知の方法等に依って変わるものである．各ＣＰＵに実装が必要となるソフトウェア機能（構造）と，その際の検討事項を表 5.2 に示す．

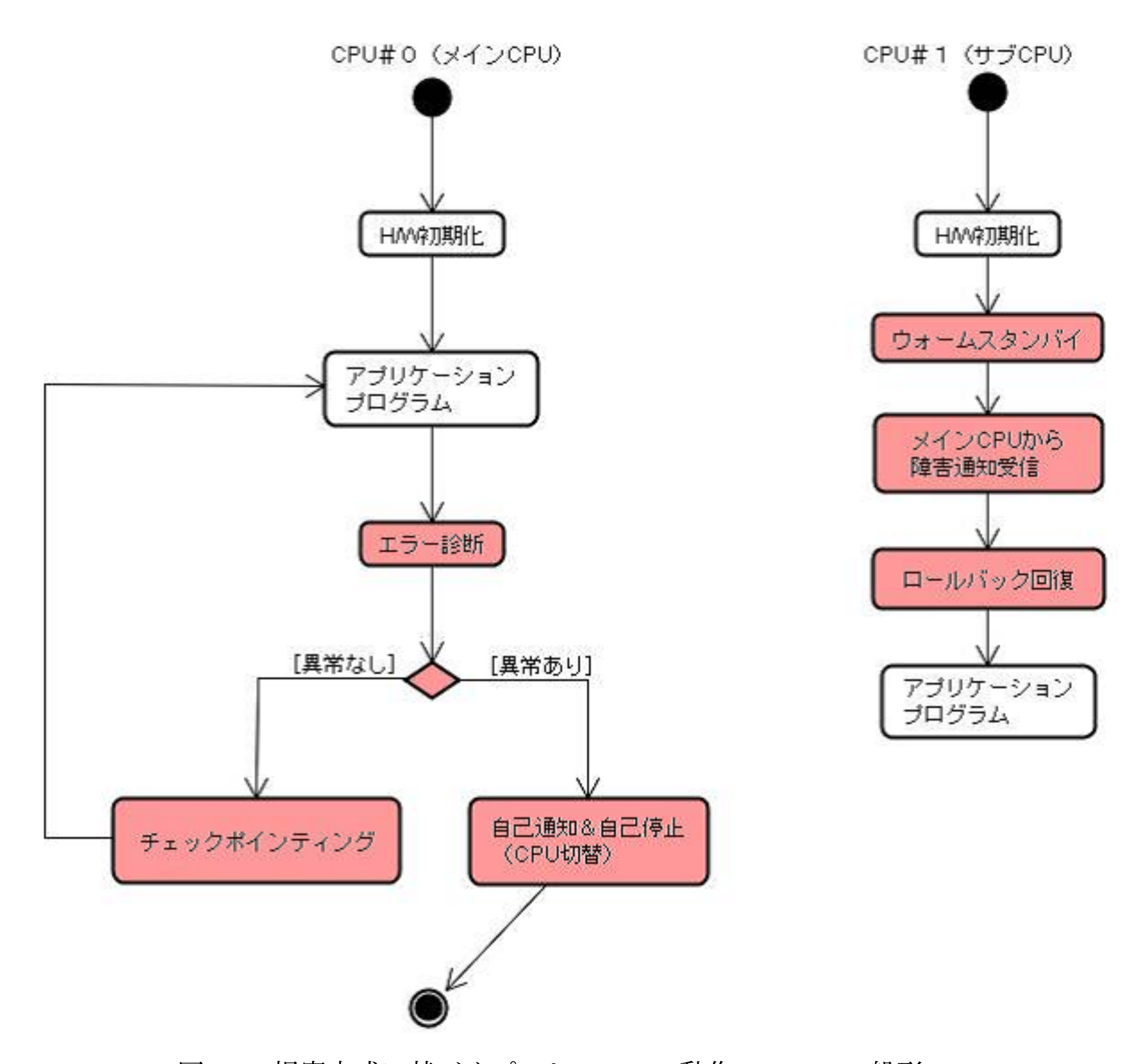


図 5.2：提案方式に基づくプロセスペアの動作フローの一般形

表 5.2：ソフトウェア機能と検討事項

実装ＣＰＵ	ソフトウェア機能（構造）	検討事項
ＣＰＵ＃０	エラー自己診断機能	・エラーの判断基準と実現方法
	ＣＰＵ切替機能	・自己通知と自己停止の実現方法
	チェックポインティング機能	・チェックポインティングのタイミング ・残す情報の種類と退避先 ・チェックポインティングの実現方法
ＣＰＵ＃１	ウォームスタンバイ構造	・実現方法
	ロールバック回復機能	・上記チェックポインティングに準ずる

1. ウォームスタンバイ構造

ＣＰＵ切替機能（後述する自己通知機能）の実現方法と合わせて，ウォームスタンバイを実現するソフトウェアの構造を提案する．メインＣＰＵからの異常発生通知（自己通知）はＣＰＵ間割り込みを利用して実現した．電源投入後，ＣＰＵや周辺モジュールの初期化等，一連のハードウェアの初期化が行われる．ハードウェアの初期化終了後，サブＣＰＵはＣＰＵ間割り込みを有効にし，アプリケーションプログラムの直前で，自ＣＰＵをスリープ状態（低消費電力モード）にする．この状態ではＣＰＵ間割り込み要求が発生するまでＣＰＵは停止するため消費電力を抑えることができる．また，ハードウェアは初期化済みであるため，システム回復に要する時間を短縮し，ユーザに見える障害時間を短くできる．このソフトウェア構造をウォームスタンバイの実現構造とした．

2. エラー自己診断機能

エラー自己診断機能には，ウォッチドッグタイマ（以降ＷＤＴと略す）を採用した構造を提案する．ＷＤＴの採用理由を下記に示す．

- 開発者にとって，より手軽に利用できる
- 検知可能なエラーの種類が多い

ＷＤＴは，カウント値がオーバーフローすると割り込みを発生させるものである．システムが正常に稼働している時には，定期的にカウント値をリセットさせる，もしくは，ＷＤＴの停止操作を行い，カウント値がオーバーフローしないようにする．正しくリセットや停止操作が行われなくなると，カウント値がオーバーフローを起こす．そのため何らかの障害が発生したと判断できる．

ＷＤＴのカウントアップ動作から割り込み要求発生までの動作は，専用ハードウェアが行う．そのため，ソフトウェアにはＷＤＴの開始と停止操作といった単純な処理以外に複雑な処理は必要ない．開発者にとって重要な事は，アプリケーションのソフトウェア構造に合わせて，開発者がＷＤＴの適切なリセットタイミングを検討し，必要に応じ，リセット操作を行うだけである．

提案するソフトウェア構造には，ＷＤＴの開始と停止時にカウント値を退避＆復帰させる構造が含まれている．これは，例えば割り込みハンドラ単位でエラー検知を行う場合，多重割り込みの発生を考慮し，ＷＤＴを仮想的に多重化し，割り込みハンドラごとに専用のＷＤＴを持たせるための構造である．

3. チェックポインティング機能＆ロールバック回復機能

これらの機能で重要な点は，チェックポインティングを行うタイミングとチェックポイントとして残す情報の種類である．エラーが発生した状態でチェックポインティングを行っ

てしまうと正常なシステム状態を回復できない危険性が生じる．最適なチェックポインティングのタイミングについてはアプリケーションのソフトウェア構造に依存するため，この節ではチェックポイントとして残す情報の種類と，その保存先について述べる（タイミングについては6章で述べる）．

まず，チェックポイント情報として変数の値を提案する．検討時の観点を下記に示す．

- 多くの場合，システムの状態は，変数を用いて表現可能であるため
- ソフトウェア開発者にとって分かりやすく，利用しやすいこと

ロールバック回復時にはチェックポイントまで処理が戻るため，アプリケーションによっては対処が必要である．そのため開発者に分かりやすく，利用しやすいという点も考慮し，変数の値を提案する（以降，チェックポイント情報という単語は，システム回復に必要な変数という意味で使用する）．その保存先に関しては，マルチコアの特徴を活かし，同じチップ内に搭載されている共有メモリに保存領域を静的に確保し，退避・復帰を行わせる構造を提案する．図5.3に退避・復帰のメモリの関係を示す．

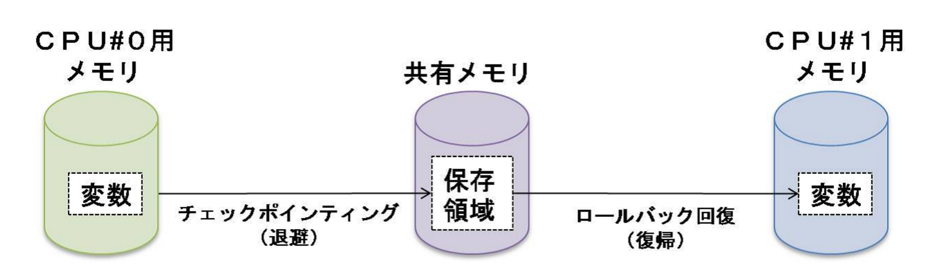


図5.3：退避・復帰のメモリ関係

4. ＣＰＵ切替機能（自己通知＆自己停止）

自己通知の実現方法についてＣＰＵ間割り込み機能を利用した構造を提案する．

一方のＣＰＵから他方のＣＰＵへ通信を行うには，共有メモリを利用した方法や，セマフォ・レジスタ（専用ハードウェア）を利用する方法等が考えられるが，今回はＣＰＵ間割り込みを利用する方法を採用し，そのソフトウェア構造を検討した．このＣＰＵ間割り込みを利用するメリットを下記に示す．

- ＣＰＵ間割り込み専用バスを使用するため，バスの競合が発生せず瞬時に割り込み要求が可能
- ＣＰＵ間割り込みコントローラのサポートにより，自己通知を実現するソフトウェア構造をコンパクトに実装可能

5.4　ＣＰＵ多重切替機能

提案したプロセスペアの方式のオプション機能として，ＣＰＵ多重切替機能を提案する．ＣＰＵ多重切替機能とは，障害が発生したＣＰＵが，ＣＰＵ切り替え後に，自ＣＰＵに対してソフトウェアリセットを行い，その後，サブＣＰＵの役割を務めることで再び障害発生に備える動作を行わせる機能である．ＣＰＵ多重切替機能の動作イメージを図5.4に示す．図中の動作を実現するソフトウェア構造を提案する．少なくとも同一のフォルトであれば動作し続けることが可能である．なお本機能をオプションと位置付けたのは，本機能の必要性はアプリケーションに依ると考えられ，本機能が無くとも，提案方式を実現できるためである．

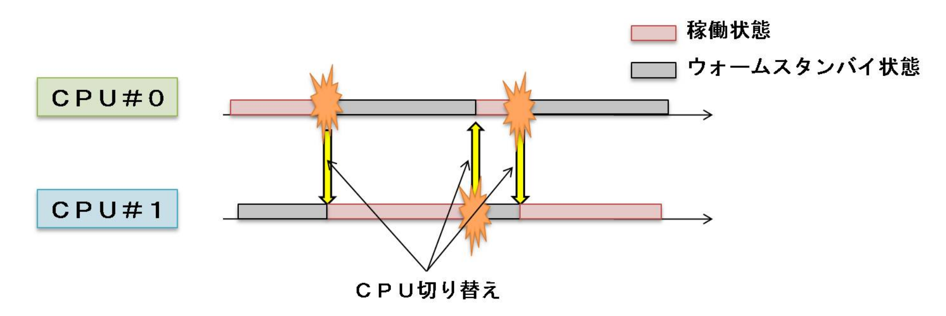


図5.4：ＣＰＵ多重切替機能の動作イメージ

5.5　提案方式の全体像

提案方式の全体像を図5.5に示す．図中の赤字が，本論文で採用し実装したソフトウェア構造である．今回実装には至らなかったが，受入れテストや出力比較等のソフトウェア構造も実装し，後述するフレームワークのコンポーネントして整備することも望まれる．

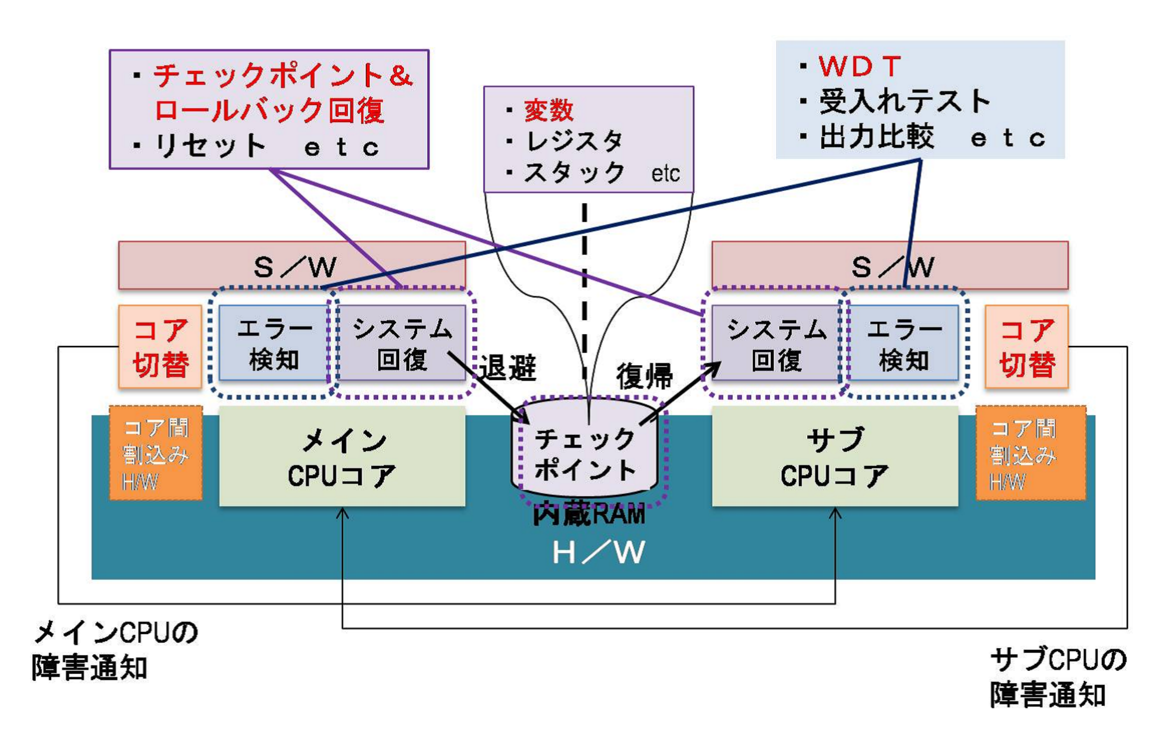


図 5.5：提案方式の全体像

第6章　フレームワークの実現

提案したマルチコア用プロセスペアの導入を支援するためのソフトウェアフレームワークを実現した．本章では，実現したフレームワークについて説明する．なお本フレームワークはＣ言語を用いて作成したが，説明の為に，本章中でのフレームワークの動作や構造に関して，オブジェクト指向に基づくイメージや概念図を用いる．より具体的な構造や，利用方法は『マルチコア用プロセスペア導入支援フレームワーク取扱説明書』（付録Ａ）および『フレームワークソースコード』（付録Ｂ）を参照されたい．

6.1　フレームワークの狙い

本フレームワークの狙いを下記に示す．

- アプリケーションを新規に開発する際の開発を効率化する
- 既存のシングルコア向けのアプリケーションを基にした開発を効率化する
- フレームワークの基本機能を応用し，マルチコアを利用したリカバリブロックの導入を支援する

本節では，上記の各狙いについて詳しく述べる．

アプリケーションを新規に開発する際の開発を効率化する

3.2 で述べたように，民生品の開発においては，その高信頼化の施策の為の開発者の負担減少も望まれる．この狙いには下記のような状況においてフレームワークが利用されることを想定し，開発の効率化を狙っている．

- マルチコア用プロセスペアを導入しておきたいアプリケーションプログラムを開発する場合
- システムの信頼性を向上させるための開発コストや開発者の負担増加を避けたい場合
- 開発期間や開発工員をシステムの高信頼化の施策に割く余裕がなく，万が一に備える機能を手軽に導入しておきたい場合

既存のシングルコア向けのアプリケーションを基にした開発を効率化する

従来からソフトウェア開発の効率化には，既存のソフトウェアを再利用する方法も一般的に行われてきた．民生品でも同様である．特に民生品の場合は既存のアプリケーションのほとんどはシングルコアチップ（シングルプロセッサ環境）向けに開発されたものである．ゆえに，本フレームワークにもシングルコア向けのアプリケーションを基にした開発を効率化する狙いがある．フレームワークを利用することが有効であろう開発状況の想定を下記に示す.

- 既存のアプリケーションは，ＲＴＯＳを使用していないソフトウェア構造であり，なるべく修正せずに高信頼化したい場合
- 既存の製品モデルのソフトウェアを再利用して，次期モデルのソフトウェアを開発する場合
- 高信頼ソフトウェアを新規に開発する余裕がない場合

フレームワークの基本機能を応用し，マルチコアを利用したリカバリブロックの導入を支援する

一般的に異なる設計に基づくソフトウェアは同一のフォルトが含まれる可能性が低いことが期待される．ソフトウェア開発者の負担は増加するが，より信頼性の高いアプリケーションの開発が期待できる．そうした技術の中でリカバリブロックの構造には，本方式を応用できる点があると考察した．本論文では図6.1 に示すリカバリブロックの構造をマルチコア用リカバリブロックと呼び，本フレームワークの応用方法を提案する．ＣＰＵコアごとに設計の異なるソフトウェアが実装されている点以外，処理の流れはプロセスペアと同じである．通常時はメインＣＰＵ（主バージョンソフトウェア）のみが動作し，受入れテストに合格しなかった場合はサブＣＰＵ（副バージョンソフトウェア）に切り替わるものである．下記に想定した応用方法の例を示す．なお本研究では，マルチコア用リカバリブロックのフレームワーク化は行わず，マルチコア用プロセスペアのフレームワークから，応用できそうな基本機能・構造を挙げ，応用例を示すだけにとどめる．

例１ アプリケーションプログラムの最重要処理の部分は２通りの設計を用意し，メインＣＰＵには設計１を，サブＣＰＵには設計２に基づくものを実装する

例２ メインＣＰＵ側にはフレームワークを用いて新規に開発したものを，サブＣＰＵ側には既存のアプリケーションをフレームワークに適用させたものを実装する

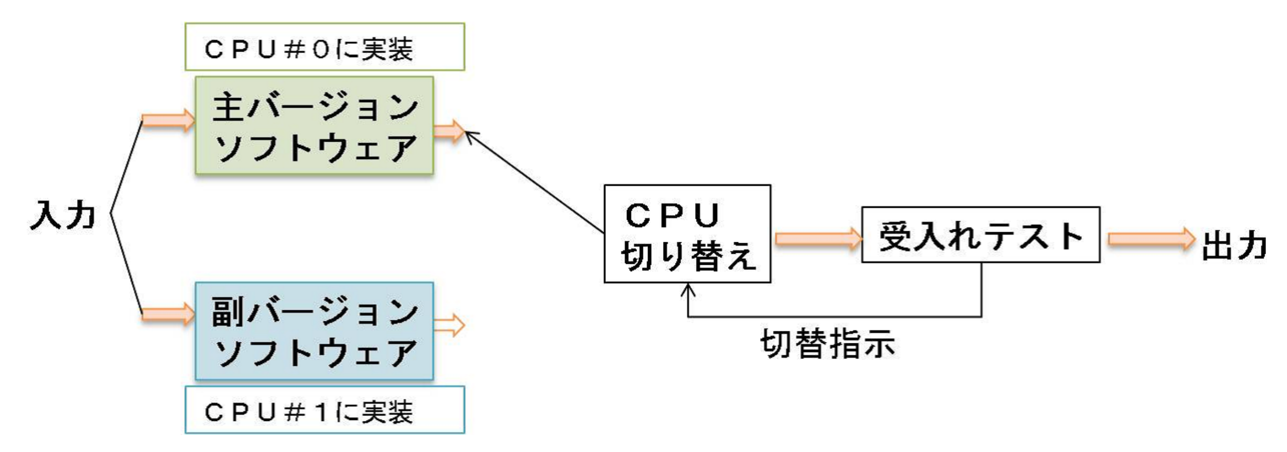


図 6.1：マルチコア用リカバリブロック

6.2　実現したフレームワークの構造

フレームワークの狙いに基づき，提案したプロセスペアのソフトウェア構造を整理し，ソフトウェアフレームワークを作成した．この節では，フレームワークの構造をオブジェクト指向の概念を用いて説明する．

フレームワークの構造を検討および実現する際は下記に示す点に注力した．

- 開発チップを他社のチップに変更した場合でも対応しやすいこと
- なるべくアプリケーションプログラムに変更を加える必要が少ない構造であること
- アプリケーション開発者にとって，利便性が損なわれないこと

本節では実現したフレームワークの構造について説明する．

ウォームスタンバイ

ウォームスタンバイ（実際に待機状態となる）までのソフトウェアの動作構造は，”ハードウェアの初期化 ”，”ＣＰＵ間割り込みの有効 ”，”メインＣＰＵからの障害通知まで待機 ”である．この一連の動作はアプリケーションに依存するものではないのでフローズンスポットと考えることができる．ただし，ハードウェア初期化方法は様々なやり方が考えられ，またハードウェアの初期化時のみに変更可能な設定もある．そのため，初期化サンプルプログラムを用意し，開発者が手を加えることも可能なホットスポットとした．ウォームスタンバイに関するフレームワーク構造をオブジェクト指向的に捉えたイメージを図 6.2 に示す．

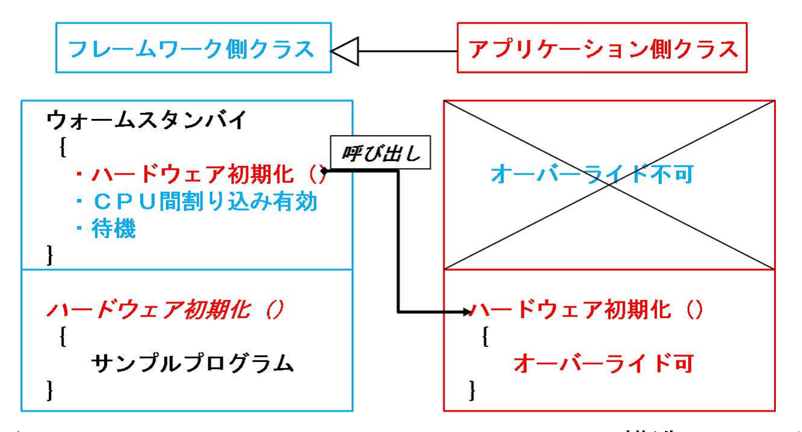


図 6.2：ウォームスタンバイのフレームワーク構造イメージ

エラー自己診断機能

アプリケーションによって，エラー検知を行いたい部分や，エラーの判定基準に合わせて，様々なエラー検知方法やソフトウェア構造が考えられる．そのため，エラー検知方法と構造を限定してしまってはフレームワークの利便性が損なわれる．

ＷＤＴには，5.3 で述べたように検知可能なエラーの幅が広く，汎用的に利用しやすいという点がある．また開発者が手軽に利用できるという点からＷＤＴを利用する構造をフレームワークのデフォルト構造としている．エラー検知をアプリケーション関数で行いたい場合は，割り込み発生時にＷＤＴを停止させる必要性が生じる可能性がある．こうしたアプリケーションごとの依存に対応することや，ＷＤＴの利便性を考慮し，ＷＤＴの操作構造をコンポーネント化し開発者に提供している．ＷＤＴに関連したフレームワークコンポーネントの一覧を表 6.1 に示す．表中のコンポーネントは開発者が自由に利用可能である．ＷＤＴに関するフレームワークの構造イメージを図 6.3 に示す．基本的な使い方は，アプリケーションプログラムのエラー検知を行いたい部分の前後に，ＷＤＴの開始と停止関数の呼び出し記述をするだけでよい．またＷＤＴをエラー検知以外に使用する場合やＷＤＴを使用しないアプリケーションにも対応可能な構造としてある．

表 6.1：ＷＤＴに関連するコンポーネント一覧

ＷＤＴ開始	・ＷＤＴを稼働させる ・アプリケーションプログラムから自由に呼び出し可能
ＷＤＴ停止	・ＷＤＴを停止させる ・アプリケーションプログラムから自由に呼び出し可能
ＷＤＴリセット	・ＷＤＴのカウント値をリセットする ・必要に応じて，アプリケーションプログラムの適切な位置から呼び出す
ＷＤＴカウント設定	・ＷＤＴのカウント値を設定する ・アプリケーション開発者の判断で，カウント値を短く，又は長く設定可能

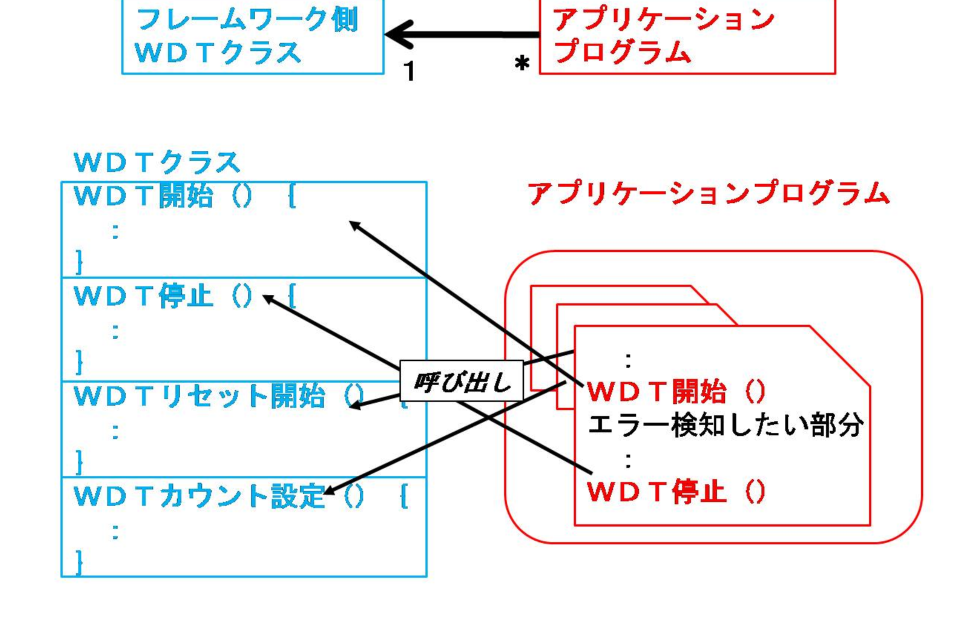


図 6.3：ＷＤＴのフレームワーク構造イメージ

チェックポイント＆ロールバック回復機能

アプリケーションによってチェックポイント情報は異なり，中には初期化されたほうがよい変数がある場合も考慮した．そのため，チェックポイント情報は開発者が静的に指定する構造となっている．また，ロールバック回復時に変数の初期化も可能な構造となっている．

チェックポイント情報のスコープ（適用範囲）も重要である．例えば，チェックポイント情報に広域変数を使用して，一元管理する構造も可能である．しかし，この構造ではアプリケーションプログラムの全てのハンドラや関数から，チェックポイント情報の参照・更新が可能となる．保守性の面からも一般的に推奨されていない．逆に局所化しすぎると

管理が大変である．ゆえに，チェックポイント情報のスコープはファイルスコープ（アプリケーションプログラムのファイルごとにチェックポイント情報を持つ）を採用した．アプリケーション開発者に，チェックポイント情報をファイルごとに静的に指定してもらう．チェックポインティングの動作は，フレームワークが全てのチェックポイント情報を，共有メモリに確保された保存領域に自動的に一括退避を行う構造となっている．ロールバック回復の動作も同様である．(図 6.4)．

チェックポイント＆ロールバック回復方式ではアプリケーションの構造に合わせて，適切なタイミングでチェックポインティングを行うことも重要であることは 5.3 で述べた．そのためチェックポインティング動作は，開発者が任意のタイミングで行えるようになっている．ロールバック回復については，メインＣＰＵからの障害通知を受けた後，アプリケーションの初期化，チェックポイント情報の復帰という動作構造となっている．この処理手順はアプリケーションに共通でありフローズンスポットとしている．アプリケーションの初期化処理（アプリケーションで使用する周辺モジュールや割り込みコントローラの設定等）は，ホットスポットとして提供している．同処理内で変数の初期化も可能である．チェックポイント＆ロールバック回復機能に関するフレームワーク構造のイメージをそれぞれ図 6.5 と図 6.6 に示す．

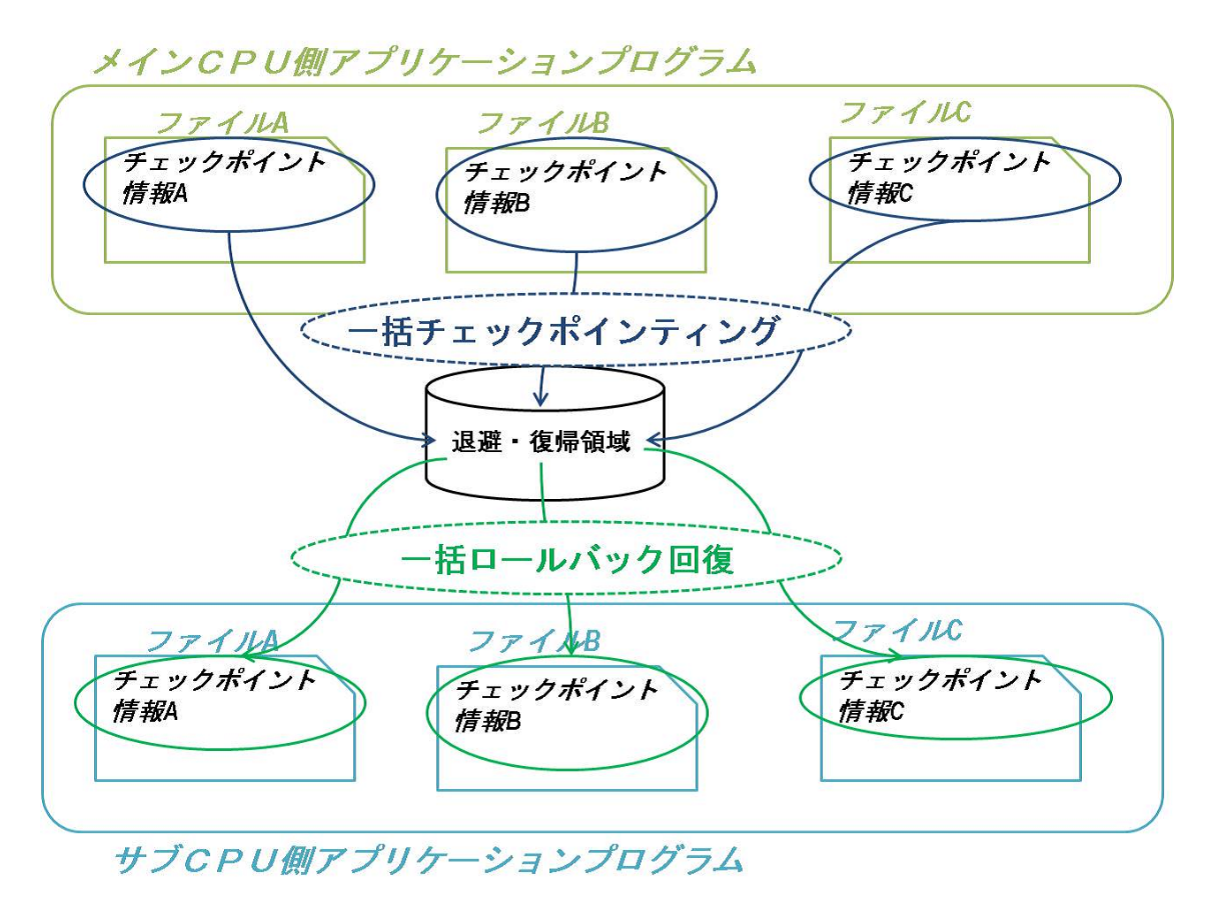


図 6.4：チェックポインティングとロールバック回復の動作イメージ

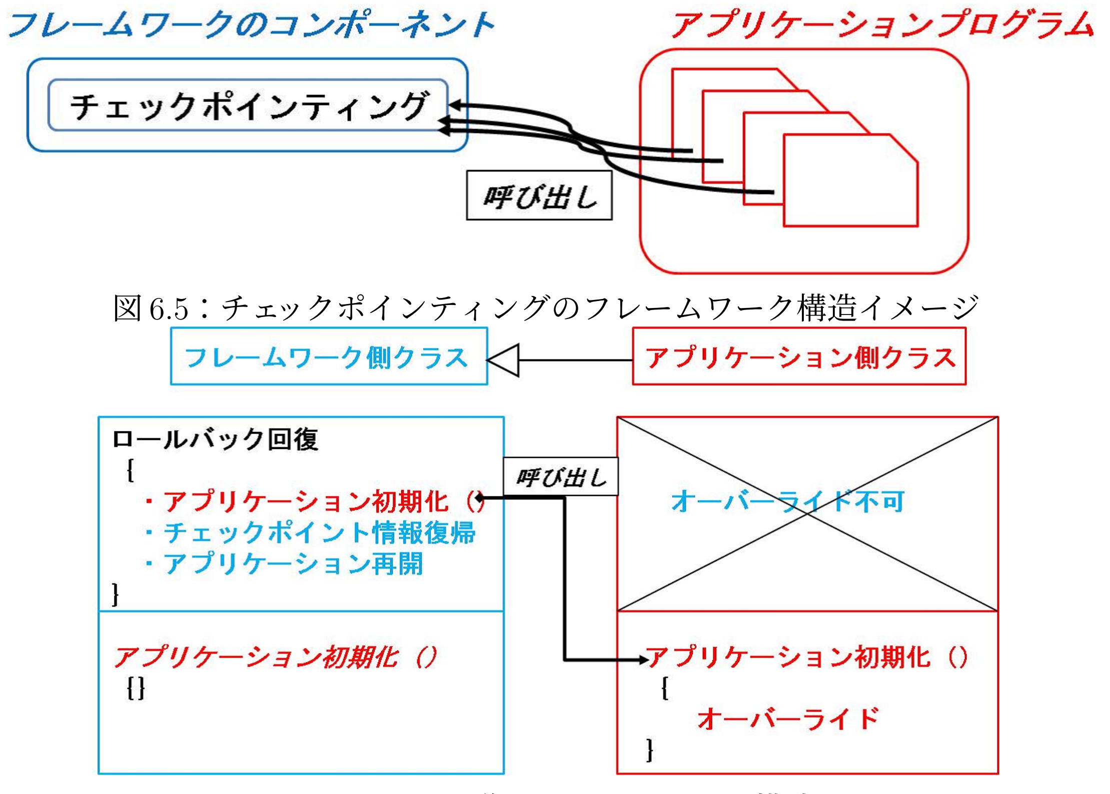


図 6.5：チェックポインティングのフレームワーク構造イメージ

図 6.6：ロールバック回復のフレームワーク構造イメージ

ＣＰＵ切替機能

ＣＰＵ切替機能の一部である自己通知は，ＣＰＵ間割り込みを利用したソフトウェア構造をフレームワーク化した．

5.1 で，従来方式であれば複数のハードウェアに依存するＣＰＵ切替機能を，マルチコア向けにコア切替で実現している点が提案方式の要点であることは述べた．マルチコア向けのフレームワークという観点からは，この機能を実現するソフトウェア構造とハードウェアとの依存が一つのチップ内に収まる点が，一つの要点である．開発チップが変更されても，対応しやすいという点が，アプリケーション開発者から見たときの，本フレームワークのメリットの一つである．この切替機能を特定のマルチコアチップに特化した構造で実現しフレームワーク化するとそのメリットが薄くなってしまう．そのため，他社のマルチコアチップにも一般的に搭載されているＣＰＵ間割り込みの専用ハードウェアを利用したソフトウェア構造のフレームワークを実現した．

障害発生後の動作は，アプリケーションに依らずＣＰＵ切り替えを行うことを前提としている．ゆえに，障害が検知されてから自己通知までの一連の動作構造をフローズンスポットとし，後述するＣＰＵ多重切替機能を考慮し，自己通知後に，ＣＰＵを停止させるかどうかはアプリケーション開発者が選択可能な構造とした．ＣＰＵ切替機能に関するフレームワーク構造のイメージを図 6.7 に示す．なお，本機能の導入に関して特に指定の無

い場合は，開発者はアプリケーションプログラムに特別な記述を追加する必要は無い．

フレームワークのクラス

ＣＰＵ切替
{
・ＣＰＵ間割り込み要求（自己通知）
・ＣＰＵ停止（）または
ＣＰＵ多重切替（）
}
静的に選択

ＣＰＵ停止（） {
:
}

ＣＰＵ多重切替（） {
:
}

図 6.7：ＣＰＵ切替機能のフレームワーク構造イメージ

6.3　フレームワークの全体像

実現したフレームワーク全体の概念モデルを図 6.8 に示す．フレームワークはＣ言語を用いて実現したが，図は概念を分かりやすく捉えるために，オブジェクト指向的に捉えたモデルとなっている．概念上ではチェックポイント情報は，アプリケーションプログラムの各クラスに属し，チェックポインティングとロールバック回復の動作は，多様性を用いて実現されるイメージである．Ｃ言語では継承や多様性といった実現方法は無いため苦労した点である．Ｃ言語での実現モデルを図 6.9 に示す．概念モデルのクラスが実現モデルのファイルに対応する．チェックポインティングとロールバック回復の動作は，各ファイルに存在するチェックポインティング関数とロールバック関数へのポインタを一元管理することで多様性を模した動作を実現した．また，フレームワークへの適用作業時にミスが入り込まないように工夫した．ホットスポットが各ソースファイルに分散し煩雑化する事を避けるために，ホットスポットを関数化し，開発者が判断しやすいようにまとめた．適用作業を支援するスクリプトも付属してある（後述）．

実現したフレームワークの全体構成のイメージ図を図 6.10 に示す．エラー検知とシステム回復機能は，フレームワークのコンポーネントとして提供する．基本的な適用作業は，アプリケーション開発者が，対象としたいエラーの種類や，開発したいアプリケーションに適したエラー検知コンポーネントとシステム回復コンポーネントを選択する．アプリケーションにコンポーネントを呼び出す記述を追加し，フレームワークに乗せることで，提案方式が導入される．今回，エラー検知コンポーネントとしてＷＤＴを実装した．システム回復コンポーネントとして，変数のチェックポイント＆ロールバック回復を実装

した．本フレームワークを適用し導入される動作の一例を図 6.11 に示す．図中の動作は，例えばエラー検知を割り込みハンドラ単位で行わせる場合の動作である．なお本フレームワークには後述のオプションも含まれている．

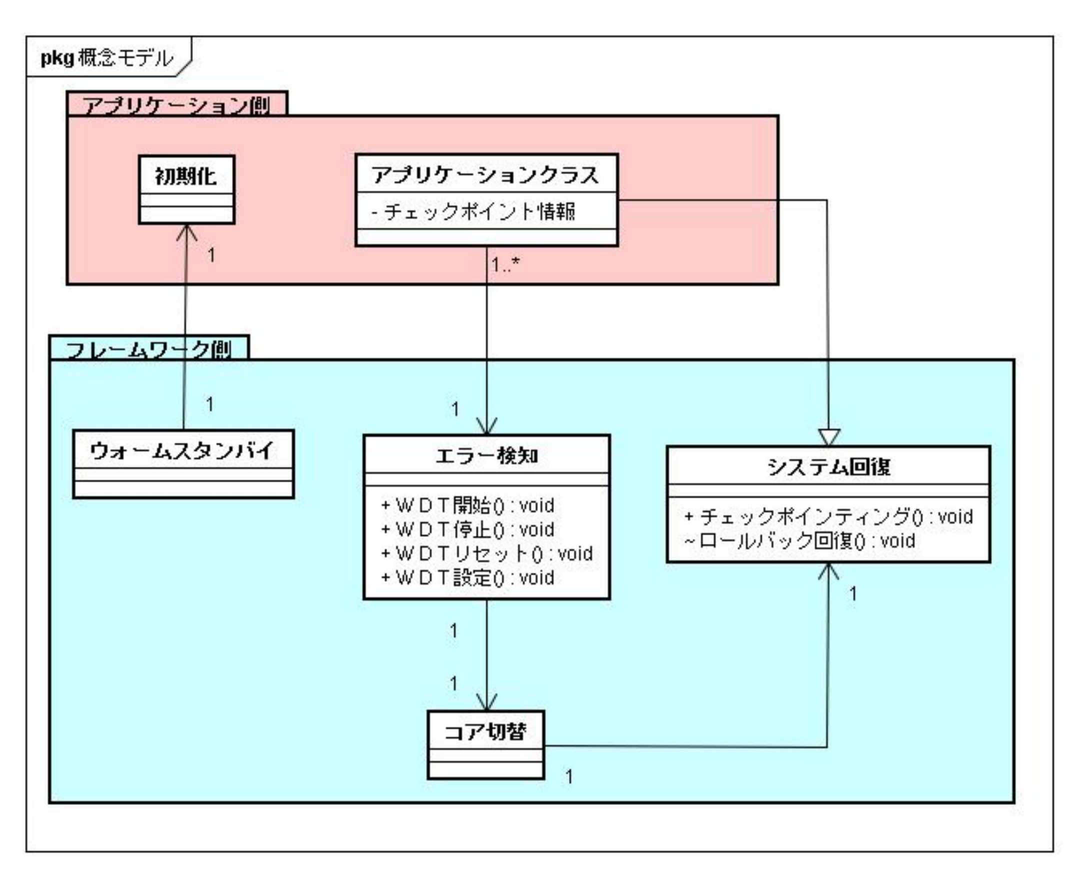


図 6.8：概念モデル

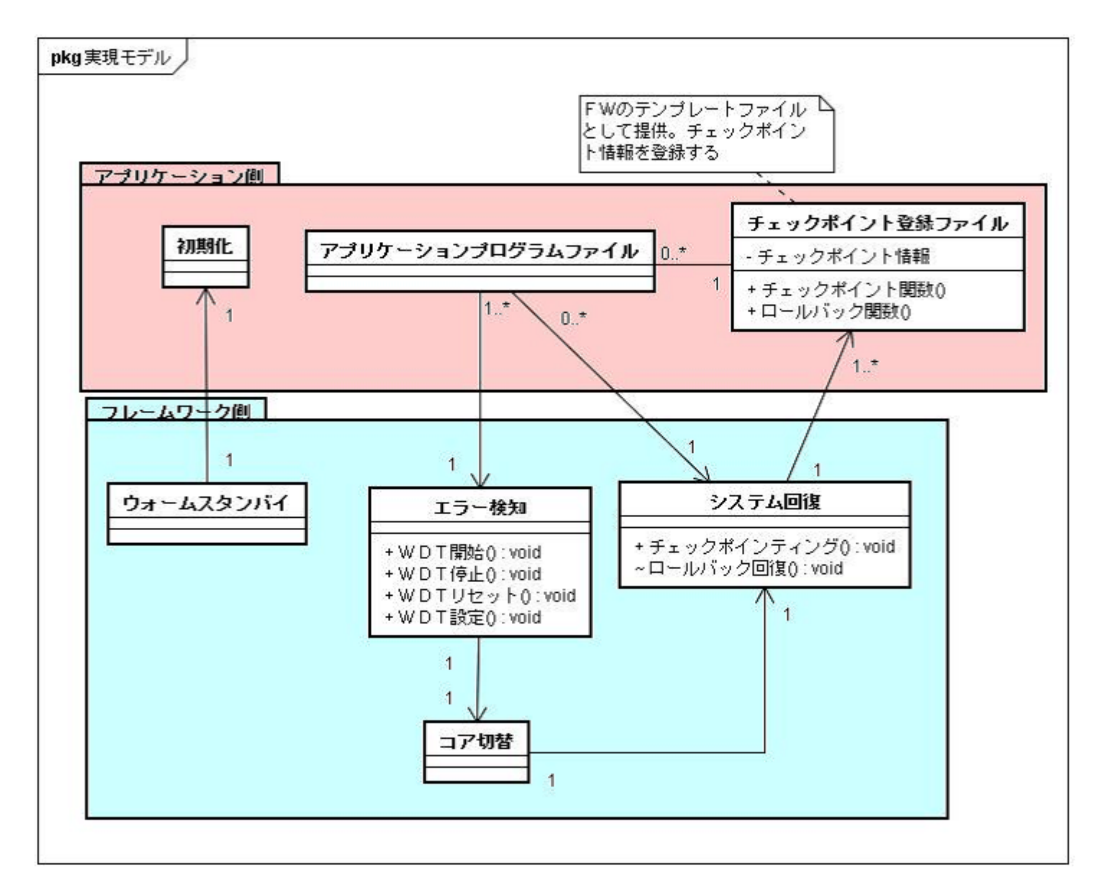


図 6.9：実現モデル

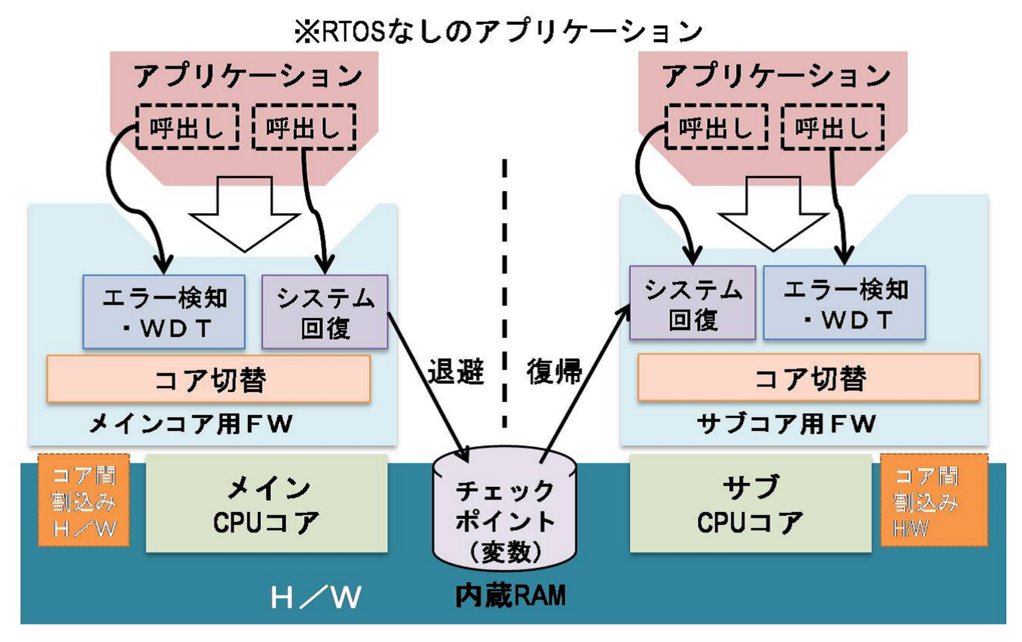


図 6.10：フレームワークの全体構成のイメージ図

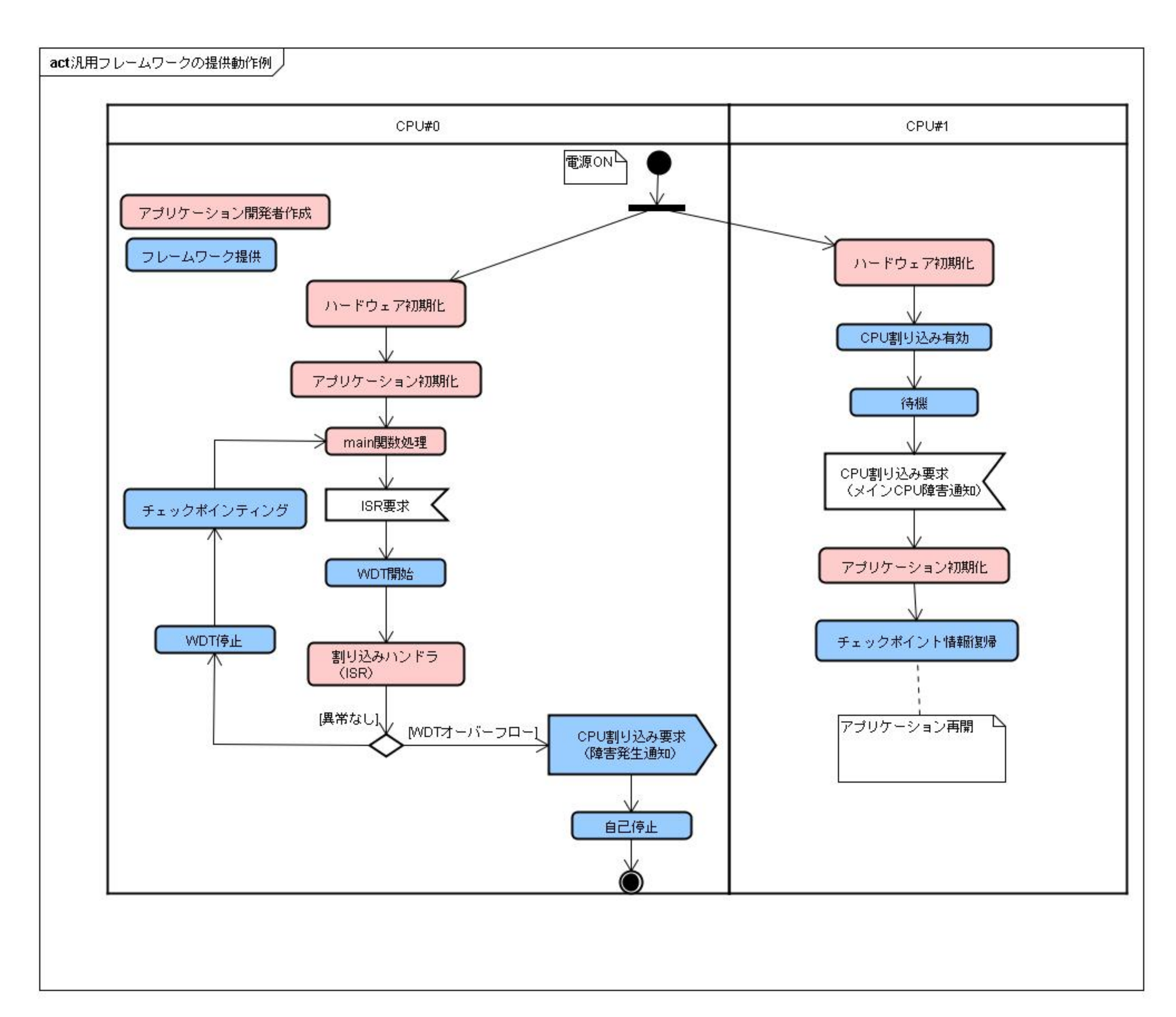


図 6.11：フレームワークを適用した時の動作の一例

フレームワークのオプション

マルチコア用プロセスペアの基本動作の他，フレームワークのオプション機能として，ＣＰＵ多重切替機能を提供する．本機能をオプションとして位置付けている理由は5.4で述べた．

ＣＰＵ多重切替機能を実現する上で，ソフトウェアフォルトが発生したＣＰＵに対して，ソフトウェアリセットを行い，次の障害発生に備えるソフトウェア構造（ウォームスタンバイと同様）が必要となる．ソフトウェアリセットの実現構造については，ハードウェアの初期化と同様に様々なやり方が考えられるので，ホットスポットとし，障害通知後，”ソフトウェアリセット ”，”ＣＰＵ間割り込み有効 ”，”再び障害が発生するまで待

機する ”までの動作手順をフローズンスポットとした．追加作成したＣＰＵ多重切替機能のフレームワーク構造のイメージを図 6.11 に示す．

また，本フレームワークを利用しやすくするために，フレームワークへの適用作業を支援するＲｕｂｙスクリプトを作成し，フレームワークに付属した（詳細は付録Ａを参照されたい）．

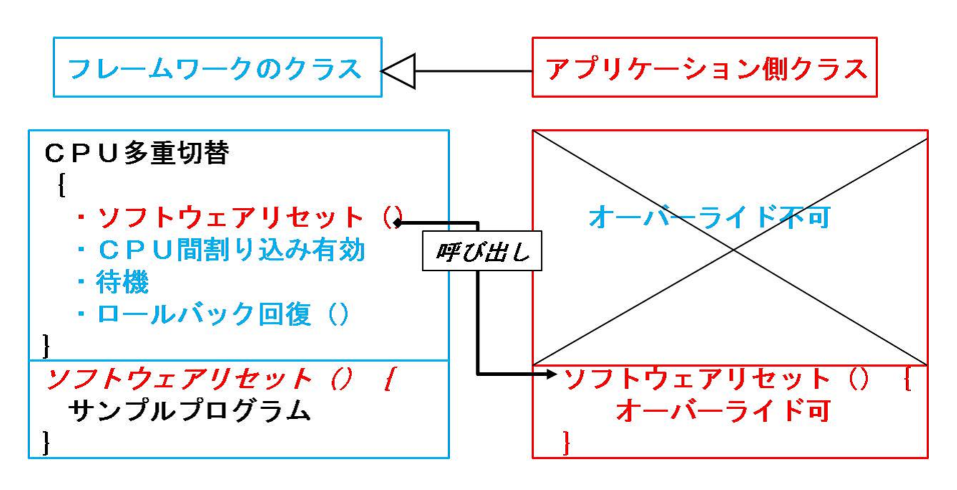


図 6.11：ＣＰＵ多重切替機能のフレームワーク構造イメージ

第7章　評価

民生品の高信頼化に対して，本アプローチの有用性を示すために下記の評価を行った．本章ではそれぞれについて説明する．

提案方式について

- 例題による実証
- 時間特性の評価

フレームワークについて

- 空間特性の評価
- 既存サンプルプログラムを用い，開発特性の評価

7.1　提案方式の評価

提案したマルチコア用プロセスペアの方式について，例題を作成し，各種機能を実証した．また，各機能のオーバーヘッド時間やシステム回復に費やされる時間を測定し，時間特性の評価を行った．本節では，各々の詳細について説明する．

7.1.1　提案方式の実証

提案したマルチコア用プロセスペアの方式を実証するための例題を作成し，例題により各種機能を実証した．例題の詳細を以下に示す．

開発環境

対象チップ：ＡＰ－ＳＨ２ＡＤ－０Ａ（ルネサス製）
開発言語　：Ｃ
開発ツール：統合開発環境ＨＥＷ

例題の想定

アプリケーションとして保温ポットを想定し，その機能を模擬した例題を作成した．想定したアプリケーションの詳細および，フォルトの想定を下記に示す.

アプリケーションのソフトウェア構造の想定

- イベント駆動型アーキテクチャが採用されている（ＲＴＯＳは使用していない）
- メイン関数と２つの周期タイマ割り込みハンドラとシリアル通信割り込みハンドラから構成される
- 多重割り込み有り

フォルトの想定

フォルトは割り込みハンドラに存在し，発生すると非稼働障害（2.1.2 を参照）を引き起こすと想定した.

アプリケーションの機能の想定

想定したアプリケーションの機能一覧を表 7.1 に示す．メイン関数と各割り込みハンドラの役割，および優先度を表 7.2 に示す．このアプリケーションに提案したマルチコア用プロセスペアを導入した．また実験用に特定の入力においてシリアル通信割り込みハンドラが無限ループ状態となるバグを作成し例題とした.

表 7.1：アプリケーションの機能一覧

機能名	内容
沸騰機能	水を沸騰させる（沸騰モード）
保温機能	水温を設定温度に保つ（保温モード）
表示機能	水温，設定温度，モード，動作ＣＰＵ（確認用）を表示する ※表示イメージ Water：95 KeepTemp：98 MODE：BOIL CPU ＃ 0
再沸騰機能	再沸騰する（沸騰モードに変更する）
保温温度変更機能	設定温度を変更する（３段階）

表 7.2：ハンドラの役割と優先度

ハンドラ	役割	優先度
タイマ割り込みハンドラ１ （ISR1）	・表示機能を実現 ・一定周期でディスプレイ表示を更新する	低
タイマ割り込みハンドラ２ （ISR2）	・モード（システム状態）に合わせて 沸騰／保温機能を実現	中
シリアル通信割り込みハンドラ （ISR3）	・シリアルポートからの入力に応じて， 設定温度およびモード（システム状態） を変更する	高

例題に導入したＷＤＴとチェックポインティングの動作

例題に導入したＷＤＴとチェックポインティングついて下記に示す．

ＷＤＴ

各割り込みハンドラの開始時と終了時にＷＤＴの開始と停止を行う．5.3 で述べたように，多重割り込みに対応するためにＷＤＴの開始処理と停止処理にＷＤＴのカウント値を退避・復帰させる仕組みを作成してある．ＷＤＴの稼働イメージを図 7.1 に示す．ハンドラごとに専用のＷＤＴのカウント値を持たせることで，各割り込みハンドラが仮想的に，専用のＷＤＴを持つイメージである．

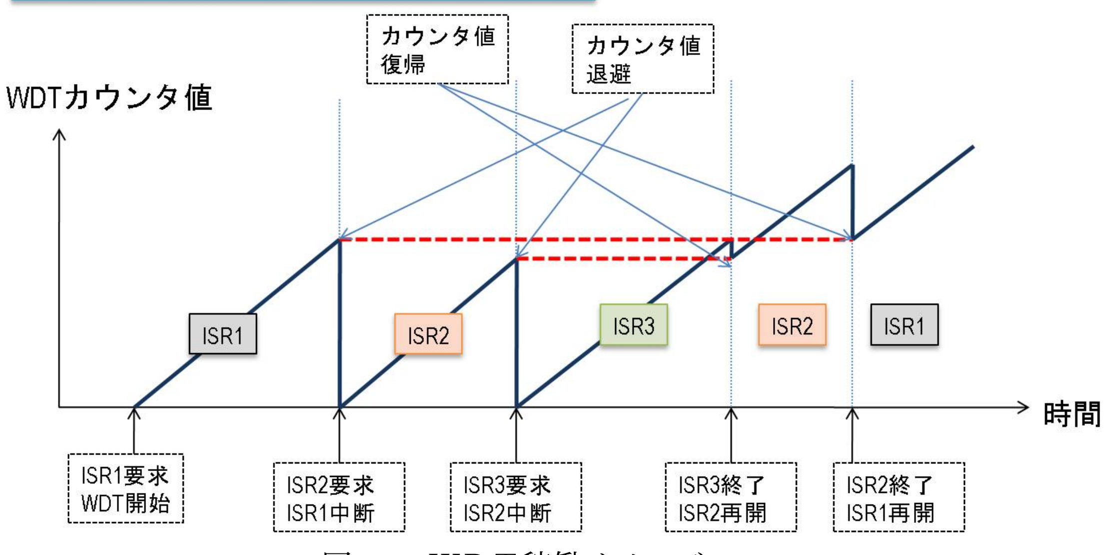


図 7.1：ＷＤＴ稼働イメージ

チェックポインティング

例題ではチェックポインティングをメイン関数で行っている．メイン関数は無限ループをしながら割り込み要求を待ち続ける．割り込み要求が入り，全ての割り込みハンドラの処理が終わると，チェックポインティングを行い，再び割り込み要求を待ち続ける．図 7.2 に動作イメージを示す．

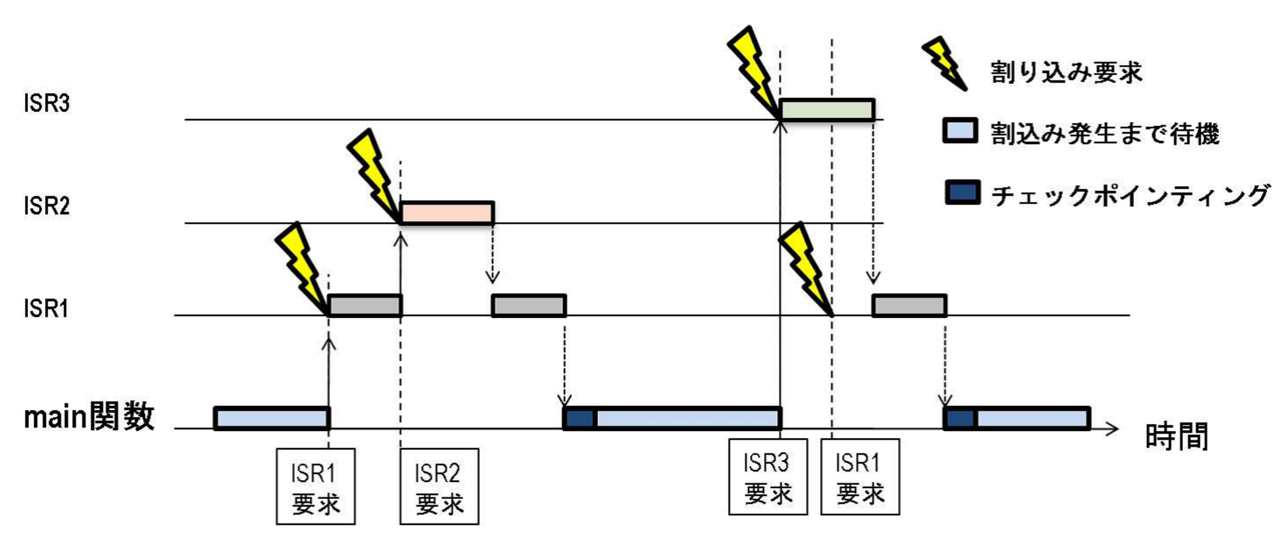


図 7.2：チェックポインティングの動作イメージ

上記の例題を用い，シリアル通信割り込みハンドラで無限ループを発生させる実験を行った．無限ループ発生後，速やかにコアが切り替わり，ロールバック回復が行われ，処理が再開される動作を確認用ディスプレイ表示で確認，およびデバッグにて，正しくチェックポイント情報が退避・復帰されていることを確認した．ディスプレイ表示の例を図 7.3 に示す．ＣＰＵコアが切り替わった後も，例題の機能は問題なく動作し，提案したマルチコア用プロセスペアの方式を実証した．

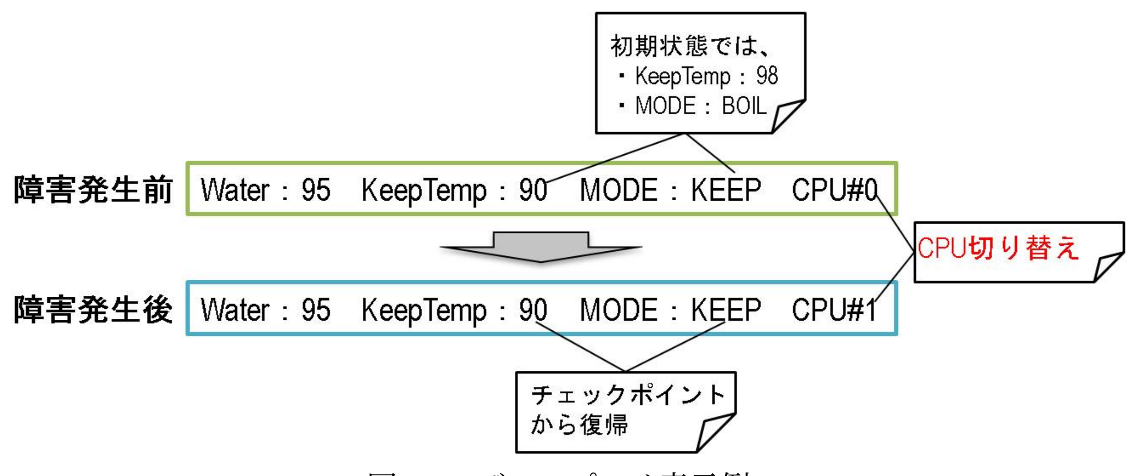


図 7.3：ディスプレイ表示例

7.1.2　時間特性の評価

評価方法

時間特性の測定環境，および評価基準を下記に示す.

測定環境

測定装置：ルネサス製Ｊ－ＴＡＧデバッガ XrossFinder
（デバッガ内蔵タイマで測定）
測定単位：ミリ秒（※デバッガ最小測定単位）

測定基準

(1) **チェックポインティングに費やす時間**

チェックポイント情報はプログラムファイルごとに存在するため，ファイル数と各ファイルに存在するチェックポイント情報のサイズを変更し相関を調べた.

(2) **システム回復時間**

メインＣＰＵのエラーが検知されてから，サブＣＰＵのロールバック回復が完了し，アプリケーションプログラムが再開されるまでの時間を測定した. ロールバック回復にかかる時間はチェックポイント情報を持つファイル数とサイズに依存するため，(1) と同様に相関を調べた.

(3) **ＷＤＴのオーバーヘッド時間**

ＷＤＴの一回分のオーバーヘッド時間は，ＷＤＴの開始と停止処理に伴う時間の和である．割り込みハンドラ単位でエラー検知を行い，全ての割り込みハンドラが専用のＷＤＴを持つと仮定する．割り込みハンドラが呼ばれるたびに，一回分のＷＤＴのオーバーヘッドが生じる．したがって，オーバーヘッド時間は多重割り込み数に比例する．今回使用したチップの割り込み要因の最大数は１４０個であるため，これらの割り込みが全て同時に発生した場合を仮定し，ＷＤＴの開始と停止処理を１４０回連続で行った時の時間をＷＤＴの最大オーバーヘッド時間とした.

測定結果

各時間の測定結果を下記に示す．各測定時間はデバッガ内蔵タイマの測定誤差を考慮し，２０回測定しその平均を採ったものである．

(1) チェックポインティングに費やす時間

測定結果を表 7.3 に示す．単位はミリ秒である．①はチェックポイントを残すファイル数（　ハンドラ数）としてアプリケーションプログラム的に妥当と考えた最大数，②はチップの最大割り込み要因数（最大割り込みハンドラ数），③はチェックポイント情報の保存領域の最大サイズ（１４０×２２８バイト　約３２Kバイト）となる組み合わせである．なおデバッガ内蔵タイマの最小測定単位上，表の緑領域と，赤領域においては優位な差が見られなかった．（緑領域：全て２ミリ秒以下，赤領域：全て３ミリ秒以下）

表 7.3：チェックポインティングに費やす時間の測定結果（ミリ秒）

	チェックポイント情報のサイズ（バイト）							
ファイル数	４	８	１６	３２	６４	１２８	１９６	２２８
１								
２								
４								
８								
１０								
２０ ※①								
１４０ ※②						4.15	5.2	5.35 ※③

※ ［緑］：全て２ミリ秒以下　［赤］：全て３ミリ秒以下

(2) システム回復時間

測定結果を表7.4に示す．表の見方は(1)と同じ．(緑領域：全て２ミリ秒以下，赤領域：全て３ミリ秒以下）

表7.4：システム回復時間の測定結果（ミリ秒）

	チェックポイント情報サイズ（バイト）							
ファイル数	４	８	１６	３２	６４	１２８	１９６	２２８
１								
２								
４								
８								
１０								
２０ ※①								
１４０ ※②						3.5	4.8	5 ※③

※ ：全て２ミリ秒以下　：全て３ミリ秒以下

(3) ＷＤＴのオーバーヘッド時間

前述した方法でＷＤＴのオーバーヘッド時間の最大時間を測定した結果，１．９ミリ秒となった（１４０回分のＷＤＴオーバーヘッド時間の和）．すなわち，１回のＷＤＴオーバーヘッド時間は０．０１ミリ秒以下といえる．

時間特性の評価

今回，測定に使用したデバッガの性能上，各機能一回当たりの正確な処理時間は計測できなかったが，いずれも１ミリ秒未満であることは明らかである．実際に，アプリケーションに障害が発生した時に，ユーザがそれをどの程度認識するかは，アプリケーションプログラムの構造（例えば，システム回復時にどれだけの手戻りが発生するか等）にも依存するため一概に評価はできない．しかしながら，マルチコア用プロセスペアの各機能の時間特性は，いずれも数ミリ秒単位であり，生活家電などの民生品にとっては概ね十分な特性であろうと考える．

7.2　フレームワークの評価

フレームワークについて，フレームワークの空間特性および開発特性の観点から評価した．

7.2.1　空間特性の評価

フレームワークの実装時のサイズを計測した．フレームワーク本体と，ハードウェアの初期化プログラムを含めC言語で実装したサイズは約１５Ｋバイトであった．本論文と同じチップを用いる場合は，このサイズにほぼ固定である．本論文で用いたチップに搭載されたメモリ（ＦＬＡＳＨＲＯＭ：１６Ｍバイト）と比較して十分小さい．ＲＯＭ以外に，両ＣＰＵから共通アクセス可能なＣＰＵ内蔵ＲＡＭに，チェックポイント情報の保存領域（最大３２Ｋバイト）を指定し使用している．サイズはアプリケーション開発者が変更可能である．メモリマップイメージを図7.4に示す．フレームワークの空間特性は，開発するアプリケーションにとって，大した制約にはならないと考える．

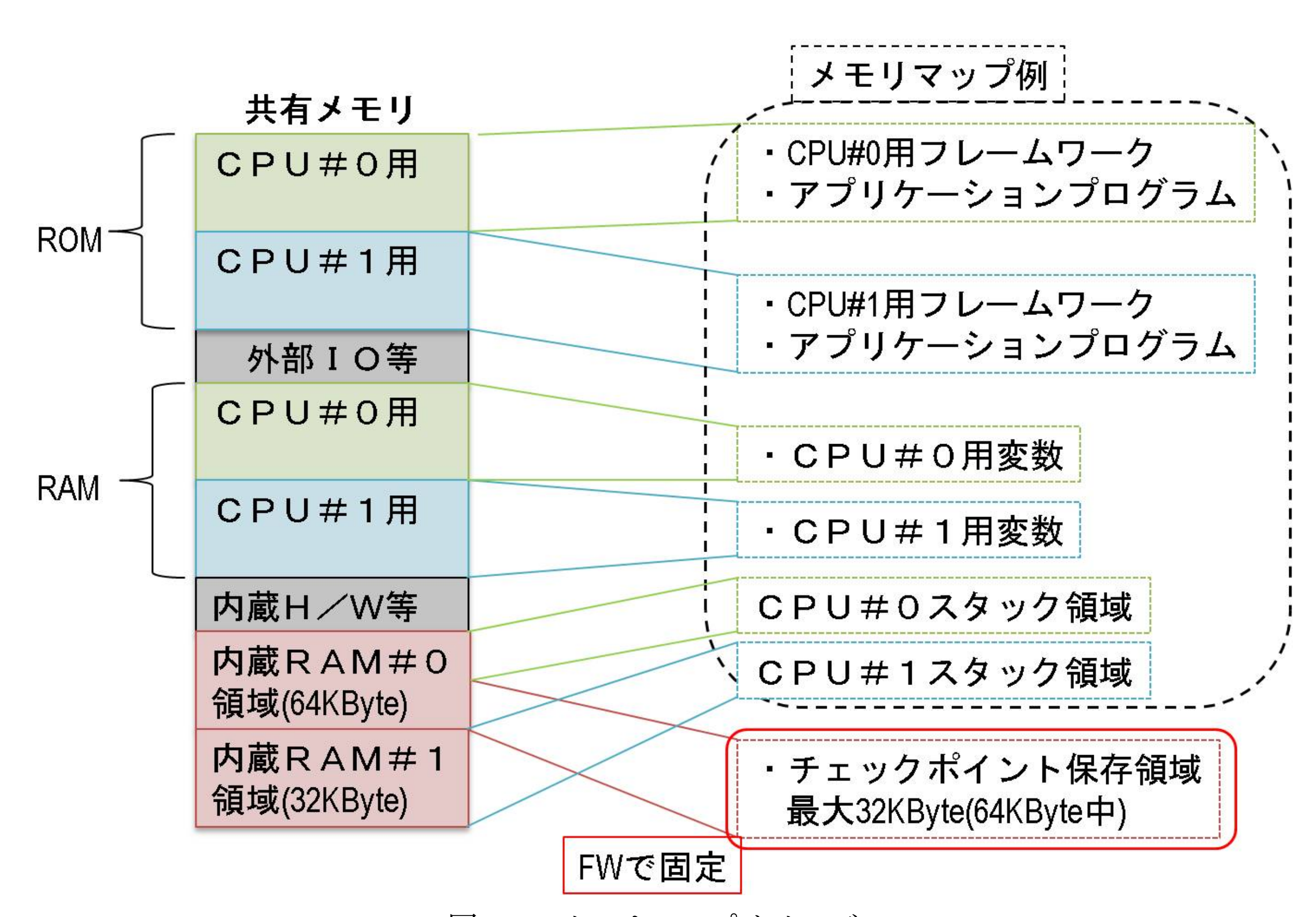


図7.4：メモリマップイメージ

7.2.2　開発特性の評価

評価方法

シングルコアチップ用に作成された既存のサンプルプログラムにフレームワークを適用した．適用したサンプルプログラムをマルチコアチップに実装し，サンプルプログラムの動作，およびマルチコア用プロセスペアの動作を確認し，フレームワークの開発特性の評価を行った．使用したサンプルプログラムと評価手順を下記に示す．

プログラム：

ルネサス製シングルコアＣＰＵチップ (AP-SH2A-0A) 用に作成されたサンプルプログラム（同社ＨＰよりＤＬ）

評価手順：

1. サンプルプログラムをフレームワークに適用し，マルチコア用プロセスペアを導入する
2. マルチコアチップに実装し，サンプルプログラムが正常に動作することを確認する
3. 意図的にエラーを発生させ，耐障害性が実現されていることを確認する
4. プロセスペアの導入作業（手順１）を，フレームワークを使用せずに行う場合と比較し，評価する

開発特性の評価

提案したマルチコア用プロセスペアの方式を，フレームワークを使用せずに導入する場合の作業について図 7.5 を用いて説明する．プロセスペアの導入には，まず，一般的に図に示すような方式の検討が必要であることは 2.2 で述べた．今回の場合，方式は決定済み（図中赤線）である．したがって，以下に示す導入作業が必要である．

(a)：ウォームスタンバイを実現するソフトウェア構造の検討と作成

(b)：エラー検知機能（自己診断）を実現するソフトウェア構造の検討と作成

(c)：チェックポイント＆ロールバック回復機能を実現するソフトウェア構造の検討と作成

(d)：ＣＰＵ切替機能（自己通知＆自己停止方式）を実現するソフトウェア構造の検討と作成

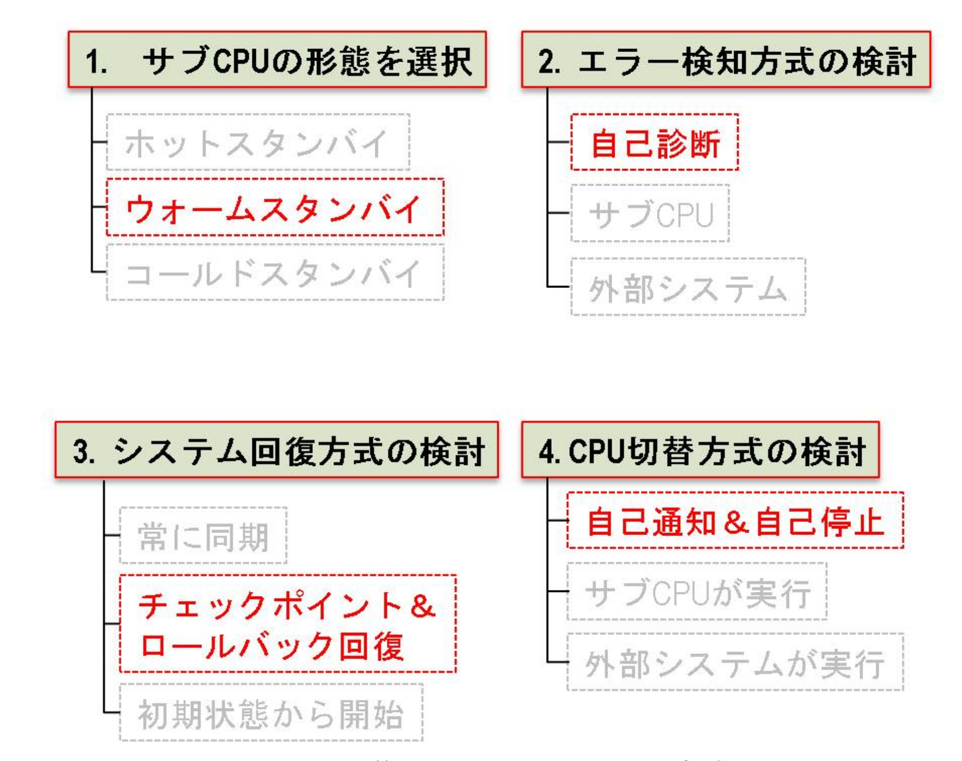


図 7.5：導入するプロセスペア方式

フレームワークを利用した場合，各導入作業は下記のようになる．

(a)：ウォームスタンバイはフレームワークのフローズンスポットで実現されるため，**導入作業は不要**である

(b)：エラー検知機能については，フレームワークの**適用方法に沿ってアプリケーションプログラムにエラー検知コンポーネント呼び出し記述を追加**する

(c)：チェックポイント＆ロールバック回復機能については，フレームワークの**適用方法に沿って，チェックポイント情報の登録作業とチェックポインティング関数呼び出し記述を追加**する

(d)：ＣＰＵ切替機能はフレームワークのフローズンスポットで実現されるため，**導入作業は不要**である

本論文では，フレームワークへの適用作業の詳細は割愛する（詳細は付録Ａを参照されたい）．既存のサンプルプログラムの大半を修正せずに再利用し，フレームワークに適用させ，マルチコア用プロセスペアを導入した．マルチコアチップに実装し，サンプルプログラムが問題なく動作すること，および耐障害性が実現されていることを確認した．マルチコア用プロセスペアの導入コストを減らし，手軽に開発できたといえる．

第8章　まとめ

この章では研究のまとめと今後の課題について述べる．

8.1　まとめ

本研究は，民生品レベルの組み込み機器を安価かつ手軽に高信頼化することを目的に行った．本論文では，マルチコアを用いたプロセスペアの方式を提案し，提案方式に沿ったアプリケーション開発を支援するためのフレームワークを実現した．提案方式および，実現したフレームワークが，民生品レベルの組み込み機器を，安価かつ手軽に高信頼化する手法の一つになりえると考える．

また従来，主として高性能化の為に用いられてきたマルチコアチップであるが，本論文での耐障害性の実証結果からも言えるように，高信頼化に活用できる可能性を示唆できたと考える．

8.2　今後の課題

今回，実現したフレームワークの今後の課題を次に示す．

- エラー検知とシステム回復コンポーネントを充実させる
- 対象フォルトや特定のアプリケーションに特化した，ドメイン特化フレームワークの構築 (図 8.1：左図)
- コア切替機能を応用したフォルトトレランス技術（例えば、マルチコア用リカバリブロック）のフレームワーク構築 (図 8.1：右図)

マルチコア用のフォルトトレランス技術のフレームワークを整備することで，より効率的な高信頼組み込み機器の開発が期待できる．



図 8.1：今後のフレームワーク

8.3 謝辞

本研究を進めるにあたり，多大なるご指導を賜りました岸知二客員教授，デファゴクサビエ准教授，青木利晃准教授にこの場を借りて感謝の辞を申し上げます．また，研究のみならず，学生生活全般にわたり，大変お世話になりました岸研究室，デファゴ研究室，青木研究室の皆様に心よりお礼申し上げます．

参考文献

[1] http://www.jpo.go.jp/shiryou/s_sonota/map/denki09/frame.htm,（2010/02/09 確認）

[2] 岸知二，野田夏子，深澤良彰：ソフトウェアアーキテクチャ，ソフトウェアテクノロジーシリーズ 4. 共立出版，2005.

[3] Avizienis,A., Laprie, J-C., Randell, B., Landwehr, C.:“ Basic Concepts and Taxonomy of Dependable and Secure Computing ”IEEE Trans. on Dependable and Secure Comupting, Vol.1, No.1, pp. 11-33, (2004).

[4] J.C.Laprie(ed.).:Dependability:Basic Concepts and Terminology. /.Spring-Verlag Wien New York，1992

[5] IEEE：Software quality characteristics and metrics.IS 9126,1991

[6] 藤原秀雄，当麻喜弘，南谷崇：フォールトトレラントシステムの構成と設計. 槇書店，1991.

[7] http://www.chugoku.meti.go.jp/event/sangakukan/h210513_3_1.pdf,（2010/02/09 確認).

[8] 米田友洋，梶原誠司，土屋達弘：ディペンダブルシステムー高信頼システム実現のための耐故障・検証・テスト技術ー. 共立出版，2005.

[9] 吉永健：『SH-2A のマルチコア化とソフトウェアの対応』，インターフェース. CQ 出版，Apr. 2008，pp.154-163.

[10] 藤倉俊幸：組み込みソフトウェアの設計＆検証. ＣＱ出版社，2006

[11] 伊藤愛，及川修一：『Gandalf VMM における Shadow Paging の設計と実装 (仮想化 (1)). 情報処理学会研究報告. 2007(36), pp.47-54

[12] http://www.nec.co.jp/techrep/ja/journal/g06/n03/060312.html,（2010/02/09 確認）

付 録A　フレームワーク取扱説明書

付 録B　フレームワークのソースコード